ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación:

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

DESARROLLO DE UNA APLICACIÓN WEB EN HTML5 PARA SMARTPHONES CON LOS DATOS DE BIBLIOTECAS PÚBLICAS PROPORCIONADOS POR OPENDATANAVARRA

Saúl Barrio Mayoral

Tutor: Marko Galarza Galarza

Pamplona, 22 de enero de 2013

I. Objetivos

En primer lugar, se quiere aclarar que varios de los objetivos de este proyecto, surgen del apartado "Líneas futuras" del PFC del alumno de la U.P.N.A, Jairo Mateos Goñi (desarrollo de un portal web optimizado para dispositivos móviles).
En éste, se proponen una serie de mejoras, como el uso de una base de datos en el portal web y el estudio de otros marcos de trabajo que hagan que la aplicación tenga una mayor velocidad de navegación.

Dicho esto, se nombran a continuación los principales objetivos de este PFC:

- Investigación de un nuevo "*framework*" o marco de trabajo, para el desarrollo de aplicaciones web para móviles. Debe sustituir a *JQuery Mobile*, cuya velocidad de navegación ha sido estudiada en proyectos anteriores y debe ser mejorada.

- Uso de un repositorio público de datos, en este caso *OpenData Navarra*, para a partir de estos datos realizar la aplicación web.

- Elaboración de una plantilla tipo combinando los dos sistemas anteriores. Esta plantilla servirá para la realización de *Web Apps* futuras.

Por lo tanto, en este proyecto se realizará la construcción, maquetación y test de una *Web App* en lenguaje *HTML5*, *CSS3* y *Javascript*. La aplicación se construirá a partir de una base de datos abierta al público y se accederá a ella mediante lenguaje *PHP* y *MySQL*.

A la hora de realizar el proyecto se tendrá en cuenta que la *Web App* tenga un diseño estético y funcional, es decir, que el interfaz sea agradable para el usuario. Todo esto se ha tenido esto en cuenta a la hora de elegir los colores y tipografías adecuados, además se ha intentando conseguir una mayor facilidad de navegación en todo tipo de dispositivos.

En la memoria se explican detalladamente los conceptos básicos para entender el desarrollo de la aplicación web, empleando un lenguaje sencillo que sea entendible para todos los usuarios.

A continuación aparece un breve estudio de los diferentes *frameworks* disponibles en el mercado, detallando sus ventajas e inconvenientes para finalmente seleccionar el que mejor se ajusta a nuestros gustos y necesidades.

Además se explican todas las tecnologías empleadas en la realización del proyecto, mostrando ejemplos del uso de cada una de ellas.

En último lugar, se sugieren una serie de posibles mejoras de la aplicación en el futuro y se detallan una serie de conclusiones sobre la *Web App*.

II. Conceptos básicos

II. I. ¿Por qué para móviles?

En este apartado se ha realizado un breve estudio de mercado, donde se muestra el avance tecnológico en los últimos años. Las conclusiones obtenidas de éste, han servido para orientar nuestra aplicación web a los dispositivos móviles.

En la actualidad, el uso generalizado de *smartphones* y *tablets* han hecho que su venta haya crecido exponencialmente.

Hace muy poco, Google lanzó un estudio en España sobre el uso de Internet en *smartphones*. Este estudio, bautizado con el nombre '*Our Mobile Planet*: *España*', muestra varias conclusiones importantes acerca de las tendencias de usuarios móviles.

Las 4 conclusiones principales que muestra el estudio son éstas:

1. Los *smartphones* se han convertido en un elemento indispensable de nuestra vida cotidiana. En mayo de 2012 la implementación de *smartphones* en España fue del 44%.

2. Los *smartphones* ayudan a los comerciantes a conectar con los consumidores. 68% ha realizado una búsqueda a través de un móvil después de ver un anuncio.

3. Los *smartphones* han cambiado la forma de comprar de los consumidores. El 82% de ellos han buscado un producto o servicio en su aparato móvil.

4. Los *smartphones* han cambiado el comportamiento del consumidor. El 58% de los usuarios móviles buscan algo en sus *smartphones* todos los días.

En otro estudio realizado se ha concluido que en España el número de aparatos móviles es mayor que el número de habitantes, esto se debe a que hay varias personas que poseen más de un terminal por distintas razones, como puede ser el trabajo.

Tras estos breves estudios, se muestran una serie de ventajas y desventajas que supone el uso de dispositivos móviles frente al ordenador a lo hora de navegar:

Ventajas

-La *movilidad* es la principal ventaja que ofrecen los dispositivos móviles, ya que su facilidad para ser llevados a cualquier lugar permite al usuario poder realizar una consulta donde quiera y en cualquier momento.

-La *conectividad* es otra de las ventajas. Actualmente la facilidad de estos dispositivos para conectarse a la red es muy grande : 3G, 4G, wifi, bluetooth etc. Todas estas tecnologías de acceso a la red permiten al usuario conectarse desde casi cualquier lugar.

-Las *diferentes funcionalidades* que ofrecen hoy en día estos dispositivos: cámara de fotos, GPS o agenda, les sitúan un escalón por encima del ordenador clásico, ya que añaden prestaciones al dispositivo haciendo todavía más práctico su uso.

Desventajas

-Las *funcionalidades* presentes en los dispositivos móviles, sin embargo, no pueden compararse a las que tiene un ordenador a día de hoy. Un ordenador cubre muchas más necesidades que un dispositivo móvil.

-El *precio* de estos móviles suele ser bastante caro, esto se debe a que ofrece muchas prestaciones en un dispositivo bastante pequeño. La relación calidad/precio de un ordenador sigue siendo mejor que en un móvil.

Una vez analizadas las ventajas y desventajas de estos dispositivos, hay que reseñar que el mercado móvil está creciendo a un ritmo vertiginoso y que su desarrollo es constante al no haber alcanzado ningún límite.

Todos estos motivos, nos han llevado a decidir que nuestra aplicación web va a estar dirigida y adaptada a los dispositivos móviles, en lugar de a los ordenadores.

II. II. ¿Qué es una Web App?

Una "*Web App*" es una aplicación software que se codifica en un lenguaje que soportan los navegadores, o dicho de una manera más sencilla, es aquella aplicación que puede ser utilizada por un usario, a través de su navegador, siempre que disponga de conexión a internet.

Es una aplicación en la que se establece una comunicación cliente/servidor, donde tanto el cliente (navegador), como el protocolo mediante el que se comunican (HTTP, FTP…) no han sido creados por el programador de aplicaciones, sino que están estandarizados.

Para entenderlo mejor, se definen en profundidad los términos cliente y servidor:

- Cliente: el cliente web es un programa con el que interacciona el usuario para solicitar a un servidor web el envío de los recursos que desea obtener mediante HTTP o FTP. Esta parte cliente de las aplicaciones web suele estar formada por el código HTML que forma la página web más una parte de código ejecutable creado en un lenguaje de script del navegador (JavaScript) o plugins para visualizar contenido multimedia, aunque esto puede generar problemas de compatibilidad con los distintos navegadores. Es decir, la misión del cliente web es interpretar las páginas HTML y los diferentes recursos que contienen. Las tecnologías más empleadas en la programación del cliente son: HTML, CSS, JavaScript y tecnologías que requieren los plugins para su perfecto funcionamiento.

- Servidor: el servidor web es un programa que espera permanentemente las solicitudes de conexión a través del protocolo HTTP o FTP por parte de los clientes web. Está formado por diferentes elementos como: documentos HTML, recursos o documentos adicionales (para emplearlos en las páginas o tenerlos a disposición para su descarga y ejecución en el cliente) y programas o scripts (ejecutados en el servidor cuando son solicitados por el navegador del cliente).

En la *figura 1* se muestra el funcionamiento de las transferencias cliente → servidor:

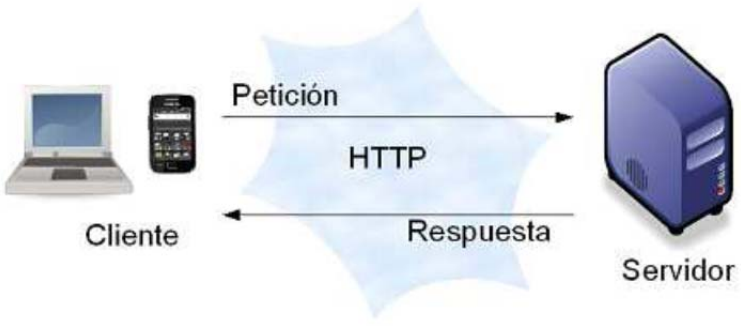

Figura 1: Transferencias Cliente- Servidor

II. II. I. Web Apps VS Native Apps

En el mercado de las aplicaciones móviles contamos con dos grupos claramente diferenciados: Las *Web Apps* mencionadas anteriormente y las *aplicaciones nativas*, o lo que es lo mismo, las que necesitan ser instaladas previamente en nuestros dispositivos. No está claro cual de ellas es mejor, esto depende del campo en el que nos centremos, para unos serán mejores las *Web Apps* y para otros las *nativas*. El cliente es el que debe decidir, basándose en su estrategia y sus necesidades, ya que para algunos campos son mejores las *Web Apps* y para otros las *nativas*, todo depende del cliente, de su estrategia a seguir y sus necesidades.

Antes de decantarnos por una de ellas, debemos tener muy en cuenta tres aspectos fundamentales:

- *No vamos a tener conexión siempre*: Este punto es muy sencillo de explicar. Si necesitamos utilizar o mostrar ciertos productos sin la necesidad de estar conectados a la red, la elección es clara, una aplicación nativa. Éstas, aunque necesitan una instalación previa en nuestro dispositivo, no necesitan disponer de conexión a internet posteriormente cada vez que las usemos.

- *Coste :*el objetivo es que nuestra aplicación web sea lo más económica posible. La elección es compleja, ya que dependiendo de los recursos que necesitemos, puede ser más rentable una aplicación nativa, aunque por lo general no suele ocurrir. Por otra parte, si tenemos en cuenta el tiempo empleado para la creación de una u otra, claramente es mejor una *Web App*, ya que un experto desarrollador puede hacer que en muy pocos días dispongas de dicha aplicación (el tiempo de creación de una nativa puede llegar a ser de varios meses).

- *Cliente:* El conocimento del tipo de cliente al que nos dirigimos es imprescindible, ya que éste puede poseer distintos dispositivos.
Teniendo esto en cuenta, serán mejores las *Web Apps*, ya que una aplicación nativa no se comporta de la misma manera en un sistema operativo que en otro (Android, iPhone...).

Una vez tenidos en cuenta estos apartados, vamos a detallar brevemente las ventajas e inconvenientes de las *Web Apps*, al tratarse de nuestra elección final.

Ventajas

- Una vez cargada la aplicación, todos los datos están a disposición del clente.
- El cliente siempre tiene acceso a la última versión de la Web App. Siempre que la cargue dispondrá de la última versión de la misma.
- Compatibilidad con todos los sistemas operativos.
- No ocupan espacio en el disco duro del cliente.

- Son compatibles con todos los sistemas operativos.
- Mayor funcionalidad con la continua actualización de los exploradores.
- Son móviles, podemos tener acceso a ellas desde cualquier dispositivo conectado a la red.

Inconvenientes

- Necesitamos disponer de conexión a la red en todo momento, para acceder a ellas y ejecutarlas. Si no estamos conectados o si se interrumpe la conexión al ejecutarlas, no podremos utilizar la *Web App*.
- El usuario sólo dispone de la última versión, no puede elegir entre cada una de las versiones usadas anteriormente.
- En el momento en que el desarrollador decida hacer desaparecer la aplicación, el cliente dejará automáticamente de poder usarla.En el caso de las nativas, el cliente siempre podrá usarla mientras la tenga instalada en su dispositivo.
- A pesar del gran avance que ha supuesto *HTML5* para la funcionalidad de las *Web Apps*, las aplicaciones nativas siguen estando un paso por delante al no verse limitadas por el navegador.

	Ventajas	**Inconvenientes**
Apps Nativas	- Interfaces más intuitivas - Mejor rendimiento - Mayor ciclo de vida de la aplicación - Presencia en la AppStore - Acceso a iAd (y otros proveedores de Ads)	- Una implementación por cada plataforma - Necesidad de personal más cualificado - Implementación más costosa
Web Apps	- Implementación multiplataforma - Puede ser desarrolladada por un programador web - Gracias a CSS3 y HTML5 pueden emularse la mayor parte de las animaciones	- Interfaces más pobres. - Poca reutilización de código. - Pesistencia muy limitada - Ciclo de vida de la aplicación más corto.

II. III. Lenguaje empleado en el desarrollo de la Web App

En esta sección se detallan los principales lenguajes empleados en la creación del código de la aplicación. Se muestra una breve definición y un claro ejemplo de cada uno de ellos.

II. III. I HTML5

El HTML5 (*HyperText Markup Language)* o lenguaje de marcado de hipertexto versión 5, es la quinta revisión o mejora del lenguaje de marcado predominante a la hora de crear páginas web, el HTML. Esta nueva versión pretende remplazar al actual HTML, corrigiendo problemas con los que los desarrolladores web se encuentran, así como rediseñar el código actualizándolo a nuevas necesidades que demanda la web de hoy en día, lo cual presenta un problema, la actualización de los navegadores y la compatibilidad con estos. Muchos de los navegadores no están actualizados, por lo que no se puede apreciar el potencial de HTML5.

En la figura 2 se muestra la compatibilidad de los navegadores web más utilizados como son: Internet Explorer, Mozilla Firefox, Safari, Google Chrome y Opera … con HTML5:

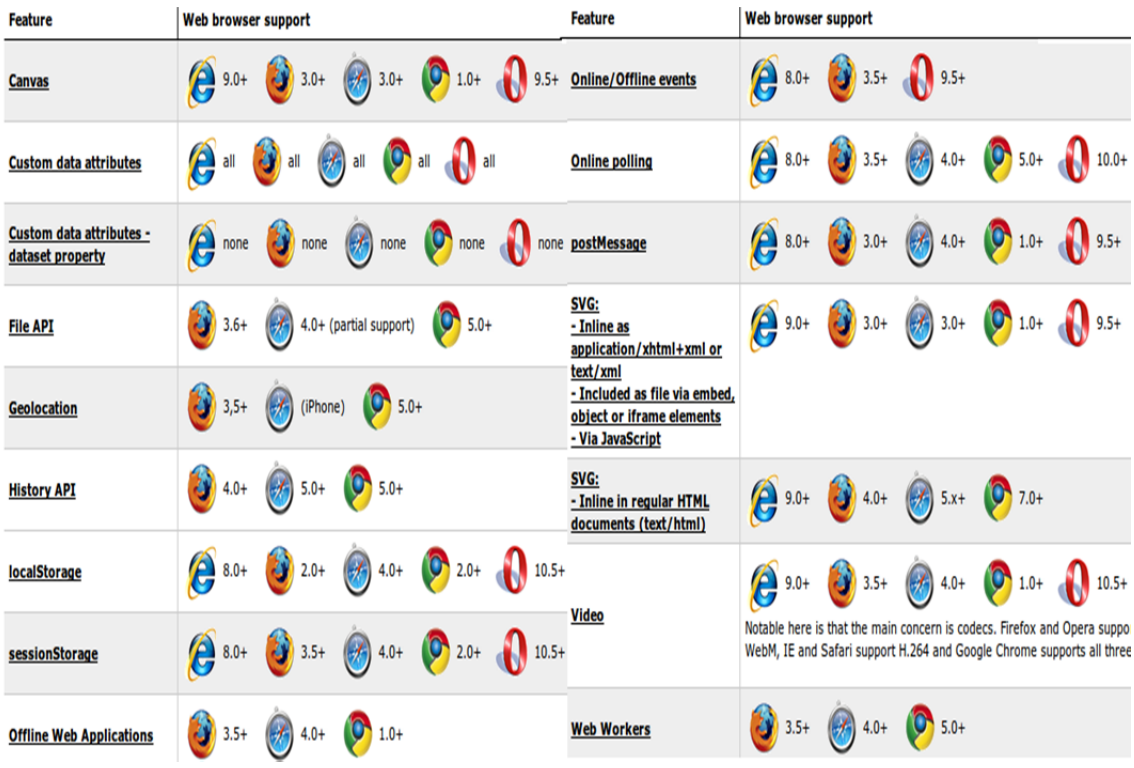

Figura 2: Compatiblidad de HTML5 en los distintos navegadores web

Las principales mejoras que introduce esta versión quinta de HTML con respecto a las anteriores son las siguientes:

- *Estructura del cuerpo:* La mayoría de las webs tienen un formato común (cabecera, pie, navegadores...), y HTML5 permite que todas estas partes representen mediante etiquetas cada una de las partes típicas de una página o aplicación web.

- *Etiquetas específicas:* Se utilizan etiquetas específicas para los contenidos enriquecidos de cada web, como lo son el audio y el video, elementos que hasta ahora se representaban todos con la misma etiqueta.

- *Introducción de Canvas:* Es un elemento que, mediante las funciones de una API (del inglés *Application Programming Interface* o Interfaz de Programación de Aplicaciones, conjunto de procedimientos o funciones que nos ofrece una biblioteca para ser utilizado por otro software) nos permitirá dibujar formas y responder a interacción del usuario, sin necesidad de tener instalado ningún plugin.

- *Aplicaciones Web offline*: Mediante un API es posible desarrollar Web Apps que funcionen sin estar conectados a Internet.

- *Geolocación:* Se pueden localizar geográficamente usuarios de las páginas web (por medio de un API también).

- *Nuevas interfaces de usuario:* Hay nuevos temas muy solicitados que se incorporan a HTML5 para su utilización.

- *Eliminación de etiquetas:* Hay etiquetas en versiones anteriores de HTML que desaparecen ya que su uso carece de provecho. Un ejemplo de esto serían las etiquetas para modificar la presentación del documento, que en HTML5 no se utilizará, y esto correrá a cargo solamente de CSS (más adelante profundizaremos más en este aspecto).

- *Mejoras en los formularios*: Posibilidad incorporación de nuevos tipos de datos sin tener que utilizar obligatoriamente *JavaScript*.

En esta nueva versión de *HTML*, se ha mejorado significativamente la estructura respecto a versiones anteriores, con unos cambios de etiqueta más funcionales.
Uno de los principales cambios, es la desaparición de las etiquetas * ,* empleadas en casi todas las secciones de versiones anteriores, dando lugar a un código confuso difícil de entender a simple vista.

A continuación se muestran los principales cambios en estas etiquetas, explicando la función de cada una de ellas para mejorar el entendimiento de este lenguaje.

- <*section*></*section*> : Representan una sección “general” dentro de un documento o aplicación. Puede contener subsecciones y si lo acompañamos de h1-h6 (formatos de títulos o textos) podemos estructurar mejor toda la página creando jerarquías de contenido, algo muy favorable para el buen posicionamiento web.

- <*article*></*article*> : El elemento de artículo representa un componente de una página que consiste en una composición autónoma en un documento o aplicación con la intención de que pueda ser repetido. Se puede utilizar en los artículos de los foros, una revista o el artículo de periódico, una entrada de un blog... es decir, en cualquier artículo independiente de contenido.

- <*aside*></*aside*> : Representa una sección de la página que abarca un contenido poco relacionado con el contenido que lo rodea, por lo que se le puede considerar un contenido independiente. Este elemento puede utilizarse para efectos tipográficos, barras laterales, elementos publicitarios, es decir, contenidos que se consideren separados del contenido principal de la página.

- <*header*></*header*> : este elemento representa un grupo de artículos introductorios o de navegación. Básicamente una cabecera para las páginas.

- <*nav*></*nav*> : representa una sección de una página que es un link a páginas externas o a partes dentro de la página.

- <*footer*></*footer*> : este recurso sirve para representar el pié de una sección, con información acerca de la página/sección que poco tiene que ver con el contenido de la página, como el autor, el copyright, año...

- <*audio*> y <*video*> : estos dos nuevos elementos mencionados antes permitirán insertar un contenido multimedia de sonido o de vídeo. Es una de las novedades más importantes e interesantes en *HTML5*, ya que permite reproducir y controlar vídeos y audio sin necesidad de plugins. Además, su comportamiento será como el de cualquier elemento nativo, con lo que podremos insertar en un video, enlaces o imágenes.

.

- <*embed*> : se emplea para contenido insertado que necesita *plugins* como el Flash. Es un elemento que ya reconocen todos los navegadores.

- <*canvas*> : Como ya hemos explicado antes, es un elemento complejo que permite que se generen gráficos al hacer dibujos en su interior. Es utilizado en Google Maps (con lo que para nuestro caso será muy importante).

En la *figura 3* se representan estas etiquetas gráficamente y en la *figura 4* su código.

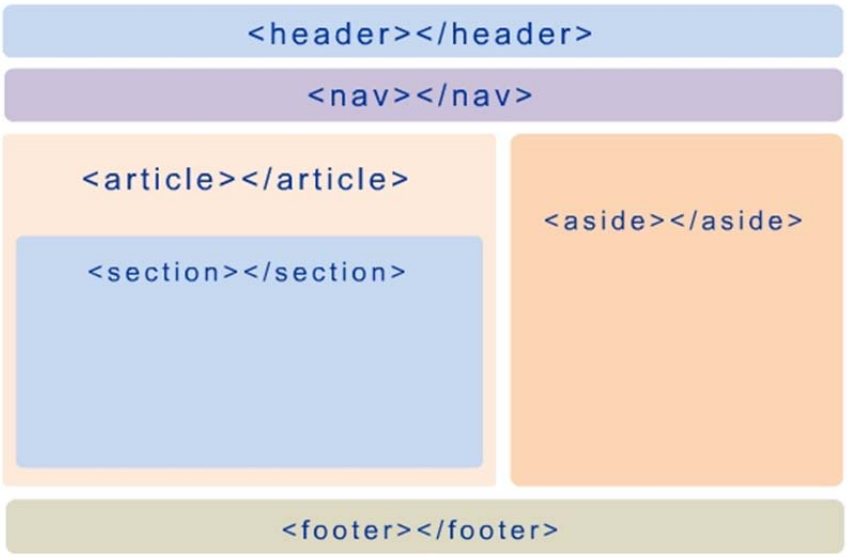

Figura 3: Estructura de las etiquetas en HTML5

Estructura.html

```
<!DOCTYPE html>
<html lang="es">
    <head>
        <meta charset="utf-8" />
    </head>
    <body>
        <header>
            <h1>Titulo tu sitio</h1>
        <nav>
            <ul>
                <li>Enlace 1</li>
                <li>Enlace 2</li>
                <li>Enlace 3</li>
                <li>Enlace 4</li>
                <li>Enlace 5</li>
            </ul>
        </nav>
        </header>
        <section>
            Aquí va todo el contenido de la Web</br>
            <article>
                <h2>Titulo del articulo</h2>
                <p>Contenido</p>
            </article>
            <aside>
                <h3>Contenido Irrelevante</h3>
                <p>Texto</p>
            </aside>
        </section>
        <footer>
            Pie de pagina, copyright, etc.
        </footer>
</html>
```

Figura 4: Ejemplo de código en HTML5

II. III. II. CSS3

CSS3 (*Cascading Style Sheets, version 3*) u hoja de estilos en cascada en su tercera versión, es un lenguaje que se utiliza para definir la presentación de un documento *HTML*. *CSS* se creó para separar el contenido de la forma, a la vez que permite a los diseñadores mantener un control mucho más preciso sobre la apariencia de las páginas.

El objetivo principal de *CSS*, separar el contenido de la forma, se cumplió ya con las primeras especificaciones del lenguaje. Sin embargo, el objetivo de ofrecer un control total a los diseñadores sobre los elementos de la página ha sido más difícil de cumplir. Las especificaciones anteriores del lenguaje tenían muchas utilidades para aplicar estilos a las páginas o aplicaciones web, pero los desarrolladores todavía continúan usando trucos diversos para conseguir efectos tan comunes y deseados como los bordes redondeados o el sombreado de elementos en la página.

La evolución ha sido necesaria: Aunque la primera versión ya significó un gran avance en el diseño de páginas web. Pero al quedar muchas otras cosas que los diseñadores deseaban hacer, y que *CSS* no especificaba en su primera versión, éstos debían hacer uso de apaños para el diseño. Lo peor de esos apaños, es que muchas veces se necesita modificar el contenido de la página, para incorporar nuevas etiquetas *HTML*, que permitan aplicar estilos de mejor manera. Dada la necesidad de cambiar el contenido, para alterar al diseño y crear estilos que esta versión de *CSS* no permitía, se estaba dando al traste con alguno de los objetivos para los que *CSS* había sido creado, que era el separar por completo el contenido de la forma, por lo que se creó una segunda versión.

Nació la segunda versión de *CSS*, o lo que es lo mismo *CSS2*, en la cual se incorporaron novedades interesantes que hoy se siguen utilizando, pero *CSS3* avanza más en la dirección de aportar más control sobre los elementos de diseño de la página.

Así, la novedad más importante que aporta *CSS3* es la incorporación de nuevos mecanismos para tener mayor control sobre el estilo con el que se muestran los elementos sin tener que recurrir a los mencionados apaños, con lo que se evita el problema de modificar el código fuente de páginas o aplicaciones web.

Para entender mejor el código de *CSS3*, en la figura 6 se muestra un ejemplo de cómo crear una animación para la parte de texto que queramos.

```
<!DOCTYPE html>
<html lang="es">
< head>
< meta charset="utf-8">
<title>GIRAR</title>

< style type="text/css">
#circulo {
background-image: url(circulo.png);
height: 100px;
position: relative;
width: 100px;
-webkit-animation:gira 3s alternate infinite;
}

@-webkit-keyframes gira
{
0% {top: 0px; left: 600px;-webkit-transform: rotate(0deg);}
50% {top: 130px; left: 200px; -webkit-transform: rotate(180deg);}
100% {top: 400px; left:600px; -webkit-transform: rotate(360deg);}
}
< /style>
< /head>
< body>
< div id="circulo"></div>
< /body>
< /html>
```

Figura 5: Ejemplo de animación en CSS3

Pero no todo son ventajas. Las mejoras que introduce *CSS3* respecto a sus versiones anteriores también suponen un problema, como ocurría con *HTML5*, ya que algunos navegadores no tienen capacidad de mostrarlas, y por lo tanto el código utilizado sería inservible. Vamos a analizar algunas de las mejoras más importantes de esta versión 3 de *CSS*:

- *Respecto a bordes*: Se pueden crear tanto bordes en las imágenes que insertamos, como modificar la curvatura, e incluso difuminarlos y crear sombras en ellos. Mención aparte tiene el color, que aunque antes ya se podía modificar, ahora disponemos una mayor gama de colores.

- *Fondos*: Anteriormente si insertábamos una imagen de fondo, esta se repetía de borde a borde sin tener en cuenta lo que el usuario quisiese, pero esto con *CSS3* ya no ocurre, ya que uno mismo indica cuántos píxeles quiere que ocupe dicha imagen. También se pueden superponer distintos fondos mediante capas.

- *Color*: Como ya hemos dicho antes, la gama de colores es mucho más amplia que en versiones anteriores de las CSS, pero además, en esta tercera versión se nos permite modificar la opacidad de los elementos que queramos.

- *Texto*: Podemos asignar todas las distintas fuentes que encontremos y crear sombras en las letras.

- *Otras mejoras:* La interfaz se puede modificar en su tamaño, se pueden crear distintos tipos de degradados, introducir columnas de texto, crear animaciones...

Como vemos hay un gran número de mejoras en *CSS3*, que además de variar el estilo y hacerlo más sencillo e intuitivo, hacen que sea una mejor herramienta para aquellas personas que quieren desarrollar una página o aplicación web.
En la figura 6 vemos un ejemplo sencillo de lo que podemos llegar a hacer con CSS3. Vemos una serie de botones con diferentes colores, letras, difuminados de color y sombras.

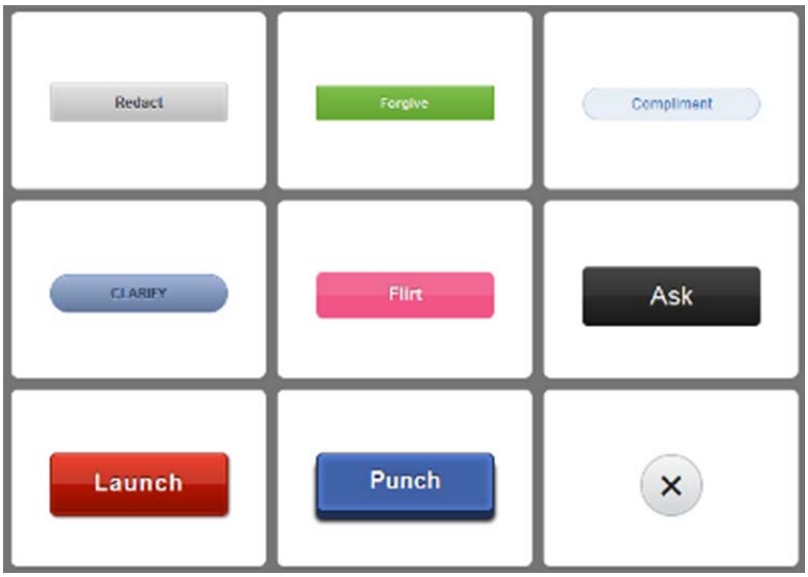

Figura 6: Ejemplo de diseño con CSS3

II. III. III. DOM

El *Modelo de Objetos del Documento (DOM)* es un API para documentos *HTML* y *XML*. Proporciona una representación estructural del documento, permitiendo la modificación de su contenido o su presentación visual. Esencialmente, comunica las páginas web con los scripts o los lenguajes de programación.

DOM es un estándar del *W3C* que nos permite construir documentos, navegar por su estructura, añadir, modificar o eliminar elementos y contenido. Se puede acceder a cualquier parte dentro un documento *HTML* o *XML*, y se puede modificar, eliminar o añadir usando el Modelo de Objetos del Documento.

Hay algunos casos en los que no podemos usar el *DOM* como son los subconjuntos internos y externos de *HTML* y *XML*.

Tras esta breve introducción, se explica de forma sencilla cómo funciona el DOM:

Cuando cargamos un documento en el navegador, se crea una estructura de objetos *(DOM)*. Ésta se puede modificar mediante el uso de *JavaScript*, alterando los contenidos y es aspecto de la página.
Son los objetos del *DOM* los que definen la forma de la ventana del navegador y todos los elementos dentro de la página (formularios, campos, párrafos, tablas …).
Con el uso de Javascrips podemos, a través del *DOM*, acceder a todos los objetos de los elementos de la página para modificar sus propiedades.
La estructura de los documentos del *DOM* tiene forma de árbol, aunque no está especificado que ésta sea la forma en la que deben implementarse.
Tampoco está definida la implementación de las relaciones entre objetos.

A continuación presentamos un sencillo ejemplo del funcionamiento del lenguaje HTML y su correspondiente estructura (en forma de árbol).

```
<html>
  <head>
    <title>Esto es un Documento</title>
  </head>
  <body>
    <h1>Esto es una cabecera</h1>
    <p id="TextoExcitante">
      Esto es un párrafo! <em>Excitante</em>!
    </p>
    <p>
      Esto también es un párrafo, pero no es ni de lejos tan excitante como el último.
    </p>
  </body>
</html>
```

Figura 7: código HTML

Como podemos ver, el documento completo está contenido en un elemento HTML. Este elemento directamente contiene otros dos: *head* y *body*. Estos se muestran en el modelo como sus hijos, y ellos apuntan hacia HTML como su padre. Y así continua, bajando a través de la jerarquía del documento, en el que cada elemento apunta a sus descendientes directos como hijos, y a su ancestro directo como su padre:

- *title* es el "hijo" de *head*.
- *body* tiene tres hijos — dos elementos p y un elemento h1.
- El elemento *p* con id="TextoExcitante" tiene un hijo, — el elemento em.
- El texto plano de los elementos (por ejemplo "Esto es un Documento!") también se representa en el DOM, como nodos de texto. No tienen hijos propios, pero apuntan a sus contenedores como su padre.

Así que la jerarquía del árbol DOM del documento HTML que hemos presentado anteriormente, se puede resumir visualmente en la Figura 8:

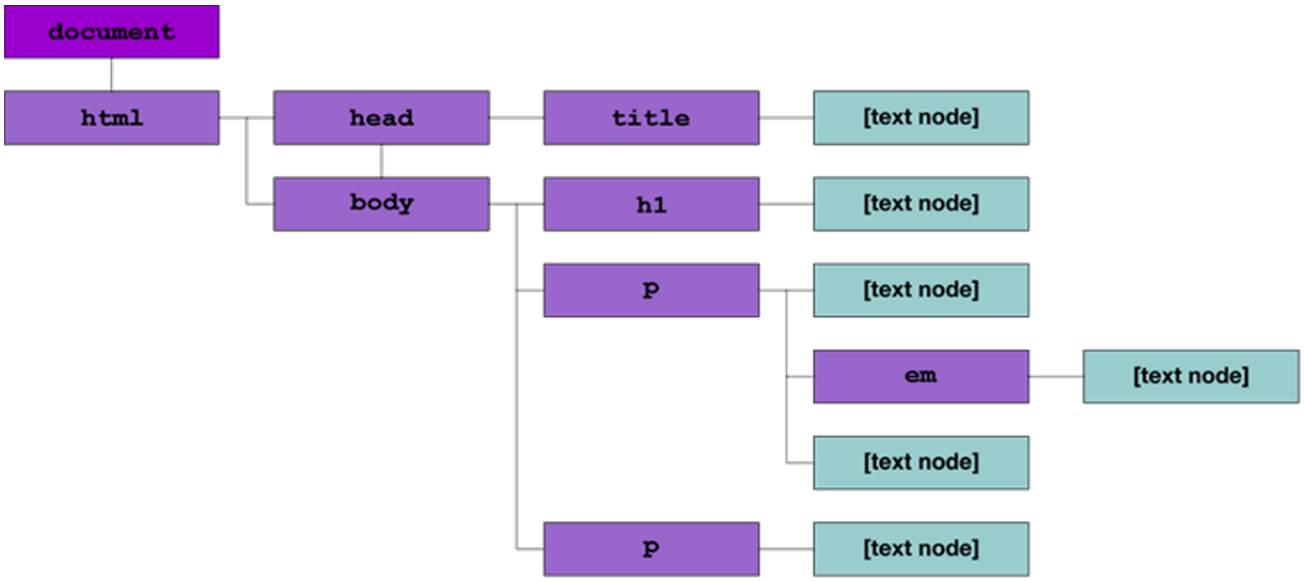

Figura 8: Jerarquía del árbol DOM de un documento HTML

Aunque en este ejemplo sólo aparecen nodos de tipo texto, hay otros tipos de nodos muy utilizados en HTML como son:

- *Documento* (*Document*): Es el nodo raíz de todos los documentos HTML y XML. Todos los demás nodos derivan de este.

- *Elemento* (*Element*): Representa los objetos definidos entre dos etiquetas de apertura y cierre (<etiqueta>..Elemento..</etiqueta>).

También son muy importante el nodo "*Atributo*" (Attr) que hace posible que podamos asignar a los objetos cierta característica o valor (color, fuente...) y "*Comentario*" (*Comment*) si deseamos mostrar algo únicamente en nuestro código fuente, sin que aparezca escrito en nuestra aplicación o página web.

A parte de los tipos de nodos, disponemos de una colección de métodos o propiedades correspondientes a todos ellos, de las cuales los más útiles son las siguientes:

- *GetElementById:* Devuelve el nodo Elemento con el identificador especificado.
- *GetElementsByTagName*: Devuelve una lista ordenada de todos los nodos elemento que tengan como nombre de etiqueta Name.
- *CreateElement* : Crea un nodo tipo Element del tipo que se especifique.
- *GetAttribute y SetAttribute*: Devuelve o añade el valor del atributo.

En nuestro caso, es decir, para el lenguaje *HTML*, también aparecen propiedades específicas como "*images*" (documentos de imagen que aparecen en un documento), "*elements*" (conjunto de elementos de un formulario), "*value*" (valor del contenido en el formulario correspondiente), "src" (lugar de origen de la imagen que queremos insertar) o "*width*" y "*height*" (anchura y altura del elemento a mostrar).

En la figura 9 se puede apreciar un pequeño *JavaScript* de cómo se utilizarían estas etiquetas:

```
<html>
 <head>
 <script type="text/javascript">
 function setSrc()
 {
 var x=document.images
 x[0].src="foto1.gif"
 }
 </script>
 </head>
 <body>
 <img src="foto2.gif" width="107" height="98">
 <form>
 <input type="button" onclick="setSrc()"
 value="Cambiar imagen">
 </form>
 </body>
 </html>
```

Figura 9: Uso de etiquetas como "src" y "value"

De esta manera, podemos tener en este proyecto, un control total sobre los contenidos que aparecerán en nuestra Web App.

II. III. IV JavaScript

Este lenguaje es la esencia de nuestro proyecto ya que se trata de un lenguaje de programación que permite interactuar dentro de la propia página.

Todos los navegadores modernos interpretan el código *JavaScript* integrado en las páginas web. Para interactuar con una página web se provee al lenguaje *JavaScript* de una implementación del *Document Object Model (DOM*) explicada anteriormente. Esto quiere decir que los programas que se escriben en este lenguaje no necesitan ser compilados ya que se pueden probar en cualquier navegador, lo que hace más llevadero el desarrollo de la aplicación web.

El principal objetivo de este lenguaje es dotar de dinamismo a las páginas web, es decir, incorporar efectos, animaciones, acciones tras ejecutar acciones en la aplicación (como pulsar un botón) y muchas más utilidades.

Javascript interactua con el código *HTML5* (aunque no es el mismo tipo de código) e incluso puede añadirse dentro de un código *HTML5*, siempre y cuando se etiquete y se defina esa parte del código como *Javascript*, de esta manera puede ser interpretada por el navegador.

```
<html>
<TITLE>Ejemplo06.htm</TITLE>

<head>
<script>
 // Ejemplo que visualiza un reloj digital.
 function Ver_Hora()
 {
   var mihora = new Date();
   var horas = mihora.getHours().toString();
   var minutos = mihora.getMinutes().toString();
   if (minutos.length == 1) minutos = "0" + minutos;
   var segundos = mihora.getSeconds().toString();
   if (segundos.length == 1) segundos = "0" + segundos;

   document.forms[0].mireloj.value = horas + " : " + minutos + " : "
+ segundos;
 }

</script>
</head>
<body>
<form>
<p align="center">
<input type="text" size="10" name="mireloj">
</p>
</form>
<script>
var r = setInterval ("Ver_Hora()",500);
</script>
</body>

</html>
```

Figura 10: ejemplo de código JavaScript

II. IV. ¿Qué es una base de datos?

Una **base de datos** es una colección de información organizada de forma que un programa de ordenador pueda seleccionar rápidamente los fragmentos de datos que necesite. Una base de datos es un sistema de archivos electrónico.

Las bases de datos tradicionales se organizan por campos, registros y archivos. Un campo es una pieza única de información; un registro es un sistema completo de campos; y un archivo es una colección de registros. Por ejemplo, una guía de teléfono es análoga a un archivo. Contiene una lista de registros, cada uno de los cuales consiste en tres campos: nombre, dirección, y número de teléfono.

A veces se utiliza DB, de *database* en inglés, para referirse a las bases de datos.

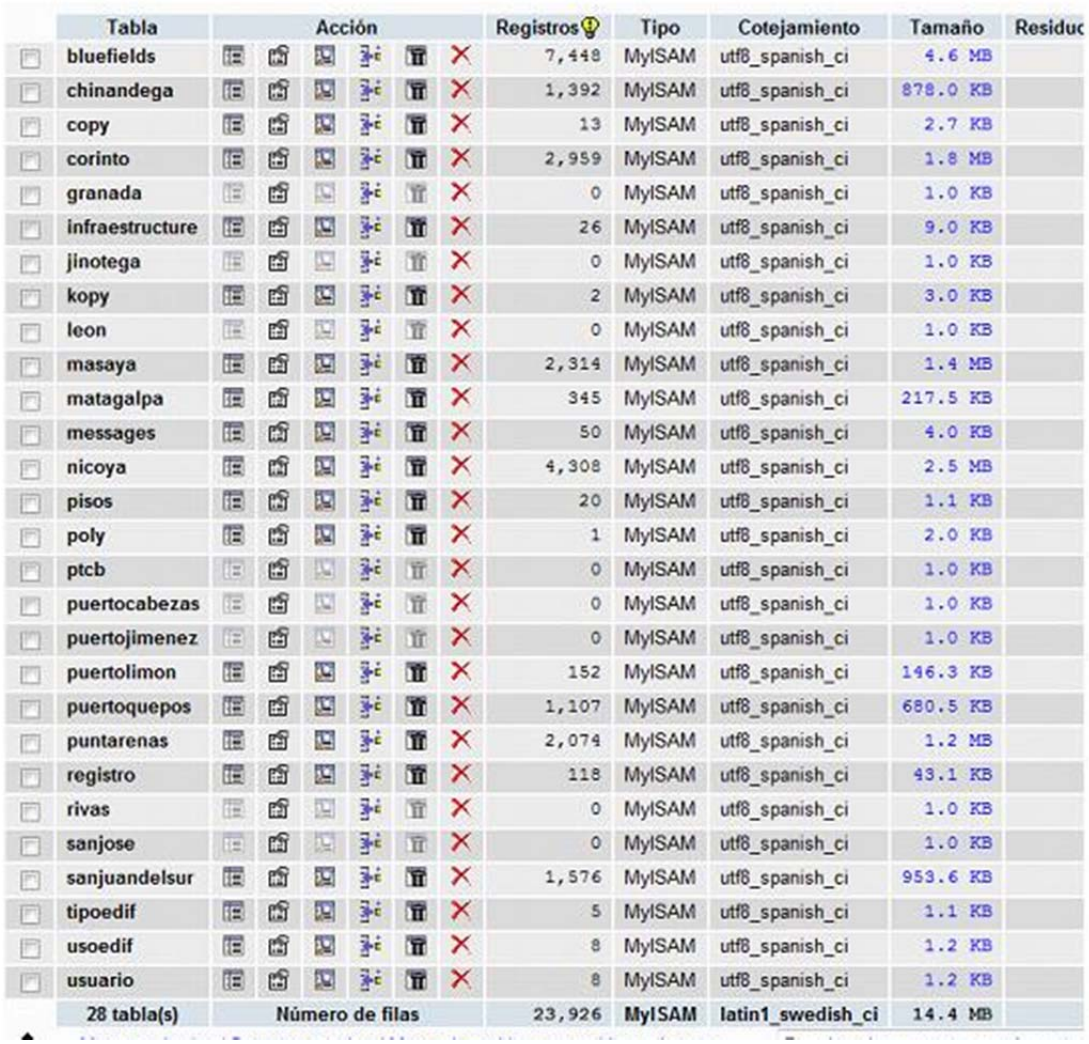

Tabla	Acción	Registros	Tipo	Cotejamiento	Tamaño	Residuc
bluefields		7,448	MyISAM	utf8_spanish_ci	4.6 MB	
chinandega		1,392	MyISAM	utf8_spanish_ci	878.0 KB	
copy		13	MyISAM	utf8_spanish_ci	2.7 KB	
corinto		2,959	MyISAM	utf8_spanish_ci	1.8 MB	
granada		0	MyISAM	utf8_spanish_ci	1.0 KB	
infraestructure		26	MyISAM	utf8_spanish_ci	9.0 KB	
jinotega		0	MyISAM	utf8_spanish_ci	1.0 KB	
kopy		2	MyISAM	utf8_spanish_ci	3.0 KB	
leon		0	MyISAM	utf8_spanish_ci	1.0 KB	
masaya		2,314	MyISAM	utf8_spanish_ci	1.4 MB	
matagalpa		345	MyISAM	utf8_spanish_ci	217.5 KB	
messages		50	MyISAM	utf8_spanish_ci	4.0 KB	
nicoya		4,308	MyISAM	utf8_spanish_ci	2.5 MB	
pisos		20	MyISAM	utf8_spanish_ci	1.1 KB	
poly		1	MyISAM	utf8_spanish_ci	2.0 KB	
ptcb		0	MyISAM	utf8_spanish_ci	1.0 KB	
puertocabezas		0	MyISAM	utf8_spanish_ci	1.0 KB	
puertojimenez		0	MyISAM	utf8_spanish_ci	1.0 KB	
puertolimon		152	MyISAM	utf8_spanish_ci	146.3 KB	
puertoquepos		1,107	MyISAM	utf8_spanish_ci	680.5 KB	
puntarenas		2,074	MyISAM	utf8_spanish_ci	1.2 MB	
registro		118	MyISAM	utf8_spanish_ci	43.1 KB	
rivas		0	MyISAM	utf8_spanish_ci	1.0 KB	
sanjose		0	MyISAM	utf8_spanish_ci	1.0 KB	
sanjuandelsur		1,576	MyISAM	utf8_spanish_ci	953.6 KB	
tipoedif		5	MyISAM	utf8_spanish_ci	1.1 KB	
usoedif		8	MyISAM	utf8_spanish_ci	1.2 KB	
usuario		8	MyISAM	utf8_spanish_ci	1.2 KB	
28 tabla(s)	Número de filas	23,926	MyISAM	latin1_swedish_ci	14.4 MB	

Figura 11: base de datos en "MySQL"

II. IV. I. OpenData Navarra

Datos abiertos (*open data* en inglés) es una filosofía y práctica que persigue que determinados datos estén disponibles de forma libre a todo el mundo, sin restricciones de copyright, patentes u otros mecanismos de control. Tiene una ética similar a otros movimientos y comunidades abiertos como el Software libre, el código abierto (open *source* en inglés) y el acceso libre (*open access* en inglés).

Los datos abiertos están centrados en material no-documental como información geográfica, el genoma, compuestos químicos, fórmulas matemáticas y científicas, datos médicos, biodiversidad... Son fuentes de datos que históricamente han estado en control de organizaciones, públicas o privadas; y cuyo acceso ha estado restringido mediante limitaciones, licencias,copyright, y patentes. Los partidarios de los datos abiertos argumentan que estas limitaciones van en contra del bien común y que estos datos tienen que ser puestos a disposición del público sin limitaciones de acceso, dado que es información que pertenece a la sociedad, como el genoma, o son datos que han sido creados por administraciones públicas (y por tanto, con los impuestos de todos), como la información geográfica o meteorológica.

En definitiva, el objetivo de poner estos datos a disposición pública, garantiza la transparencia de la entidad que los ponga a disposición.

En *OpenData Navarra*, es de agradecer que una institución pública tan importante como la Administración, aporte estos datos de forma libre, garantizando la transparencia en el sector público.

Pero si hay algo que nos ha gustado de esta iniciativa es el mercado que se crea paralelamente en torno a ella.

Para las empresas supone una gran oportunidad de crear mercado, al poder crear nuevas herramientas y servicios digitales innovadores.
Para los usuarios supone una gran ventaja el disponer de estos servicios y para las otras empresas significa que esta tecnología acarreará una reducción de costes al facilitar el manejo y distribución de datos.

En noviembre 2011 se publica una primera versión del Catalogo de Productos. Es un sistema de información que permite catalogar los productos software de la Dirección General de Gobierno Abierto y Nuevas Tecnologías que dan servicio a distintas unidades de Gobierno de Navarra. El proceso de catalogación no está terminado, por lo

que se actualizará; desde el enlace se puede descargar el documento en pdf, y en la ficha de datos el *dataset*, es decir, los datos en diversos formatos como *XML*, *CSV* o *JSON*.

El Gobierno de Navarra está fomentando el uso de este nuevo proyecto abierto al público mediante concursos y premios. En 2011 *OpenData* realizó un concurso sobre aplicaciones móviles utilizando estos datos de la Comunidad Foral con el fin de promover y promocionar su uso.

Por estos motivos decidimos emplear alguno de los datos de este nuevo proyecto impulsado por el Gobierno de Navarra, ya que de esta manera los datos son oficiales y a su vez apoyamos una iniciativa que favorece a todo ciudadano de la Comunidad Foral.

Figura 12: Ejemplo de datos públicos en OpenData Navarra

III. Estudio de los diferentes frameworks

Un marco de trabajo o '*framework*' es una estructura estandarizada de tecnologías que sirve de base para la realización de proyectos, haciendo que éstos tengan una fácil y sencilla distribución. Suele incluir bibliotecas, soportes de programas, y un lenguaje interpretado, entre otras herramientas, lo que facilita el desarrollo y la unión de los diferentes componentes de un proyecto.

En el caso que nos interesa se emplea un *framework* para desarrollar aplicaciones móviles que posee una sencilla estructura y que también dispone de unas bibliotecas propias.

Todo esto facilita la creación de una *Web App*. Es similar una plantilla vacía de una web a la cual el desarrollador va configurando y añadiendo los elementos según sus gustos y sus necesidades.

La gran ventaja de usar estos *frameworks* es la rapidez en la creación de las aplicaciones, ya que el programador no tiene que escribir todo el código ya que existe una capa previamente creada y el programador únicamente la modifica a su gusto.Con este método se reutilizan muchos componentes software.

El framework más conocido y empleado en proyectos similares ha sido *jQuery*, que se ha ido desarrollando y mejorando desde 2005.

Uno de los problemas que presenta el *framework* es que nos obliga a descargar una librería más desde nuestro sitio Web lo que implica un aumento del tiempo de carga de nuestras páginas. La librería de interfaz de usuario puede llegar a ser demasiado pesada aunque no tenemos que incluirla completa (e incluso podemos no utilizarla en absoluto).

El segundo problema es que es muy lento, lo que hace imposible el rápido funcionamiento de una aplicación como la que se va a crear, que entre otras cosas pretende ser rápida y sencilla en su manejo.

Es por esto que se decidió no utilizar *jQuery* e investigar nuevos frameworks más recientes que todavía no están tan desarrollados o estandarizados para de esta manera poder aprender el uso de un nuevo y eficiente framework que suponga una clara alternativa a *jQuery* , cumpliendo con todas las funcionalidades requeridas por la aplicación .

En este PFC se ha realizado un estudio previo de 3 de los frameworks más importantes a día de hoy, se hará una descripción detallada de cada uno de ellos, mostrando sus ventajas e inconvenientes, su compatibilidad y un ejemplo de su implementación. Por último se justificará la elección de uno de ellos para la realización de nuestro PFC.

III. I. LungoJS

III. I. I Características

LunjoJS es un framework de desarrollo para aplicaciones móviles web, que sigue el estándar *HTML5/CSS3* para la creación del código fuente. Está desarrollado por *TapQuo*, que es una empresa española afincada en Bilbao, especializada en desarrollo web y sobre todo en tecnologías *JavaScript*.

El desarrollo del framework estuvo motivado cuando el CEO de la compañía, Javier Jiménez Villar, se puso a investigar sobre tecnologías de desarrollo para dispositivos móviles, una vez consiguió su primer *iPhone* (que no llegó a venderse en España). Al desarrollar primeramente en nativo (*Objective-C*) y darse cuenta de que en este caso su desarrollo no sería compatible con distintos dispositivos (para *Android* debería usar su *API Java*, para Microsoft sus propias tecnologías...), decidió volver a sus raíces y probar el desarrollo de *Web Apps*. En ese momento empezaban a sonar las tecnologías *HTML5* y *CSS3*, y al disponerse de un estándar de futuro para estos desarrollos se abrieron las opciones.

Al investigar los dos frameworks dominadores en el desarrollo de *Web Apps* de estos momentos (*Sencha y jQuery Mobile*), se encontró con algunas dificultades e inconvenientes que le hicieron plantearse la implementación de su propio framework.

Estos inconvenientes son derivados de la filosofía que han adoptado los dos grandes desarrollos, al adaptar sus bibliotecas de desarrollo para aplicaciones de escritorio al entorno móvil. Esto hace que las bibliotecas sean relativamente pesadas al utilizar mucho código que en realidad no se necesita en el mundo móvil. Además los eventos táctiles de estas pantallas son distintos a los eventos de ratón que se utilizan en los entornos de escritorio, y no siempre están bien resueltos.

Otro de los aspectos que quería conseguir es tener un entorno realmente *HTML5 /CSS3* con el que trabajar. Según el mantra que el propio Javier se impuso para el desarrollo de Lungo: "*LungoJS solo dará soporte a dispositivos que den soporte real aHTML5/CSS3/JavaScript*".

En el caso de *jQuery Mobile,* debido a que intenta ser compatible con la mayoría de dispositivos del mercado, aunque estén comenzando a ser obsoletos, lastra su

evolución debido a que la limitación la pone el dispositivo con menos características que se ha decidido incluir en el grado-A de compatibilidad.

La filosofía de Lungo es totalmente distinta: el framework implementará las características que estén descritas en el estándar *HTML5*, y por lo tanto sólo dará soporte a aquellos dispositivos que tengan una compatibilidad real con este estándar.

Presumiblemente, cada vez los sistemas van a tener una compatibilidad con HTML5 más depurada, por lo que a priori parece una buena estrategia de crecimiento.

La licencia de este *framework* es *GPL v3* para desarrollo de aplicaciones (es decir, software libre). Si se quiere usar de forma empresarial, se debe contratar una licencia comercial de pago.

A la hora de comenzar con un desarrollo en este framework, en los tutoriales y en la misma descarga de las bibliotecas, se nos recomienda usar una serie de ficheros *JavaScript* para organizar el desarrollo. Por supuesto, se puede usar otra organización si lo creemos necesario, pero la propuesta es realmente cómoda de usar y efectiva:

- **app.js**: Para inicializar la aplicación.
- **data.js**: Para las llamadas al sistema de almacenamiento webSQL.
- **events.js**: Para la organización de eventos.
- **services.js**: Para las llamadas a servicios remotos (ejemplo, llamadas AJAX).
- **view.js**: Para la definición de vistas, mediante plantillas (templates).

III. I. II Ventajas e Inconvenientes

A continuación se muestran las ventajas e inconvenientes que hemos encontrado en este framework a la hora de realizar *Web Apps*.

Ventajas

- Creación sencilla de aplicaciones web para dispositivos *iOS (Apple), Android y Blackberry*

- Desarrollo de aplicación *HTML5* con estructura semántica en todo el proyecto, comenzando por la plantilla en *HTML*, apoyándose en los estilos *CSS* y terminando con su *API Javascript*.

• Utiliza una librería para el manejo del DOM de la página extremadamente ligera, llamada *QuoJS* y desarrollada también por *TapQuo*, de manera similar a cómo trabaja *jQuery*.

• Aprovecha las capacidades de los móviles actuales.

• Implementación sencilla de características *HTML5* como *WebSQL,* Orientación, Conexión...

• Captura eventos táctiles como *swipe, tap, doble tap...*

• Puede extenderse la funcionalidad del *frameworks* mediante *plug-ins*, que en este entorno se llaman *Sugars*.

• Diseño totalmente personalizable.

• Permite distribuir las aplicaciones tanto en sitios web como en las distintas *stores* (*Google Play, App Market de Apple...*).

Inconvenientes

• A pesar de ser muy parecido a *JQueryMobile*, la cantidad de *Web Apps, plugins* … disponibles, así como la participación de desarrolladores en foros, es infinitamente menor (está poco extendido).

• La ayuda proporcionada por su creador, Javier Jiménez Villar es nula, llegando a ser insultante para los usuarios que crean aplicaciones con Lungo.

• Su funcionamiento está limitado al navegador *GoogleChrome*, tanto en *Internet Explorer* como en *Mozilla Firefox* no ofrece un buen comportamiento.

III. I. III Compatibilidad

Como ya se ha comentado anteriormente las Web Apps creadas con "LungoJS solo son soportadas en dispositivos que den soporte real a HTML5/CSS3/JavaScript.

Por lo tanto, a día de hoy, su compatibilidad está asegurada con los sistemas más utilizados (*Blackberry, Android e iOS*) y *Google Chrome* como navegador de escritorio.

Desde septiembre de 2012 (inicio del PFC) a enero de 2013 ha habido una constante mejora en Lungo, pasando por las versiones: *1.1.1, 1.1.2*, 1.2 (usada en este PFC) y la recientemente publicada *2.0.*

En este tiempo se ha acentuado el desarrollo de los sistemas operativos compatibles con Lungo, mientras que el resto de sistemas operativos se han quedado obsoletos.

No cabe duda que la visión de futuro de *LungoJS* es increíble y el hecho de estar desarrollado con muy pocos medios y apenas un par de personas es algo muy meritorio.

III. I. IV Ejemplo

Para inicializar la aplicación, se debe indicar en el fichero app.js, mediante las siguientes líneas de código:

```
var App = (function(lng, undefined) {

    lng.App.init({
        name: 'LungoJS Test Source',
        version: '1.2'
    });
```

En cuanto a la definición del marcado en la página *HTML*, se utiliza la nomenclatura recomendada para las páginas en *HTML5*, es decir, se definen como secciones que en su interior contendrán cabeceras, pies de página y artículos.

Una sección puede contener varios artículos. Si pensamos en la organización que pueda tener un blog, las secciones de la página pueden ser por ejemplo una lista con el archivo por meses de las entradas, y otra sería el sitio en el que se indica el texto de las entradas. Los artículos serían cada una de las entradas del blog.

En *LungoJS*, definiríamos la sección principal mediante el código indicado a continuación:

```
<section id="main">
    <header data-back="home blue" data-title="Título de la sección"></header>

    <footer class="toolbar"></footer>

    <!-- content -->
</section>
```

En esta sección se definen además la cabecera, en la que se indica el botón de atrás, que al estar marcado semánticamente, se define con el icono de *HOME* y de color azul, mediante el atributo *data-back*. También se indica el título de la sección, mediante el atributo *data-title*.

Para el *footer*, se le indica en la clase que se va a comportar como una barra de herramientas, por lo que estaría preparada para que indicáramos botones en su interior.

Además de cabeceras y pies de sección, en su interior se pueden indicar artículos, tal y como se muestra en el ejemplo siguiente:

```
<section id="main">
    <!-- header -->

    <!-- footer -->

    <article id="first_article">
        <!-- content -->
    </article>

    <article id="second_article">
        <!-- content -->
    </article>

</section>
```

El artículo se puede configurar para mostrarse como una lista indicándolo mediante atributos en el elemento *article*, tal y como se puede ver:

```
<article id="first_article" class="list" data-search="blue">
    <!-- content -->
    <ul>
        <li>Primer elemento de la lista</li>
        <li>Segundo elemento de la lista</li>
    </ul>

</article>
```

En la figura 13 se puede observar el resultado en el navegador:

Figura 13: Ejemplo de visualización de una aplicación en LungoJS

III. II. Sencha Touch

III. II. I Características

Este segundo framework que hemos estudiado se basa en librerías *JavaScript* y fue creado por la empresa *Sencha*, resultante de la unión de de *ExtJs*, *jQTouch* y *Raphaël*.

Sencha Touch es un *framework* que permite desarrollar aplicaciones web para dispositivos móviles táctiles, dando prácticamente la sensación de que son aplicaciones nativas de los sistemas operativos de los dispositivos.

Fue el primer *framework* construido a partir de estándares web, y diseñado para aprovechar específicamente la potencia, flexibilidad y optimización del lenguaje *HTML5*, *CSS* y *JavaScript*. Como ya sabemos, la utilización de *HTML5* le ha dotado la posibilidad de ofrecer componentes como el audio o el vídeo, además de un proxy local para almacenar información incluso en estado "*offline"* (sin conectar a la red).

Respecto a su hoja de estilos (*CSS*), este framework ha implementado una capa de estilos independiente de la resolución que tengan los dispositivos móviles. Gracias a una combinación de tamaños *CSS3*, ha logrado adaptar los componentes de una interfaz de usuarios a la resolución de cada dispositivo (deben ser compatibles, por supuesto). Además, se aprovecha de una tecnología diferente de los otros frameworks (*SASS*) que se basa en implementar una capa de *CSS* en la que se añaden variables y funciones, con lo que se puede cambiar la presentación de una aplicación totalmente.

Sencha proporciona una librería con múltiples *widgets* como pestañas, formularios, listas... Todo ello pudiendo utilizar temas creados por el usuario.

En cuanto a la manipulación del DOM, hay que destacar el gran tamaño de su librería, con varias colecciones, paquetes y clases para facilitar la reutilización de código.
Todo se basa en heredar y codificar nuestros propios objetos, diseños y componentes con códigos que ya existen. A continuación, hacemos referencia a las clases principales del Framework siguiendo su jerarquía de herencia o el árbol DOM.
Mediante el siguiente script se consigue llamar al DOM:

```
<script type="text/javascript">

Ext.onReady(function() { //se usa para inicializar los componentes en el momento adecuado
  Ext.BLANK_IMAGE_URL = "common/ext/resources/images/default/s.gif";
  Ext.QuickTips.init();

});

</script>
```

Figura 14: ejemplo de script para llamar al DOM en Sencha

"*Ext.onReady*" es la función más importante, sirve para inicializar los componentes justo cuando el DOM ya ha sido cargado, lo cual significa que cuando el DOM esté listo ejecutará la acción.
Además de ésta, otra función interesante es "*renderTo*", que recibe un elemento DOM para renderizar el componente dentro de este elemento
Las demás funciones, como es lógico, son propias del lenguaje HTML ("*title*", "*width*", "*height*"...).

Sencha Touch presenta grandes características para decantarnos por este framework a la hora de realizar aplicaciones dinámicas. Cabe destacar, que presenta un conjunto de datos muy potente y robusto, que permite solicitar y obtener datos a a través de diferentes fuentes como *"Ajax"*. Además permite unir esos datos a plantillas o listas *HTML*.

III. II. II Ventajas e Inconvenientes

A continuación se muestran las ventajas e inconvenientes que presenta este *framework* a la hora de realizar *Web Apps*.

Ventajas

- Nos permite crear aplicaciones complejas utilizando componentes predefinidos.

- Evita el problema de tener que validar el código para que funcione bien en cada uno de los navegadores (*Firefox, IE, Safari, Opera* etc.) al estar estandarizado.

- Una ventaja fácil de apreciar es el funcionamiento de sus ventanas flotantes, en este aspecto es superior a cualquier otro *framework*.

- Relación entre Cliente-Servidor balanceado: Se distribuye la carga de procesamiento entre ellos, permitiendo que el servidor pueda atender más clientes al mismo tiempo, es decir, que se repartan el uso de recursos entre esas dos partes.

- Eficiencia de la red: Disminuye el tráfico en la red al contar las aplicaciones con la posibilidad de elegir qué datos desea transmitir al servidor y viceversa.

- Comunicación asíncrona. En este tipo de aplicación el motor de render puede comunicarse con el servidor sin necesidad de estar sujeta a un clic o una acción del usuario, es decir, se puede cargar información sin que el cliente se de cuenta.

- Hay una gran cantidad de componentes muy diversos, por lo que podemos crear el diseño de una Web App de muchas formas diferentes con un código similar.

- Presenta un Soporte para geolocalización y mapas, incluyendo un componente que implementa el *API de Google Maps*.

Inconvenientes

- Al ser el primer *framework* que apareció, los errores o bugs presentes fueron corregidos sin poder apoyarse en otros.

- Es uno de los *framewoks* más pesado, es decir, que los recursos de sus aplicaciones ocupan más que los de la competencia, detalle por el cual, muchos usarios pueden renegar de *Sencha Touch* y decantarse por otros.

- Su principal inconveniente es que el mayor peso de las aplicaciones hace que sean lentas y se produzcan sobrecargas al usarlas.

III. II. III Compatibilidad

Otra de las grandes desventajas de este framework es su compatibilidad con los dispositivos móviles. Se ha creado la versión 2 de *Sencha Touch*, que aunque tiene muchas variantes y mejoras que su primera versión, todavía está desarrollada en *HTML5* para muy pocos sistemas operativos de móviles. Únicamente es compatible con Iphone (todos sus productos), *Android* (versiones desde 2.3 hacia adelante) y algunos dispositivos de *BlackBerry* (un grupo muy reducido). Esto hace que esté un nivel por debajo de *LungoJS*.

III. II. IV Ejemplo

En este apartado se mostrará un ejemplo del funcionamiento de *Sencha Touch*, con su código y el ejemplo en el navegador, detallando las partes más importantes de éste.

Las aplicaciones en *Sencha Touch* funcionan mejor cuando siguen estructuras simples que este mismo framework proporciona para que el código sea sencillo y entendible. El primer paso es establecer la estructura. Inicialmente lo único que se necesitan son dos archivos y copiar la carpeta de *Sencha Touch*:

- *app.js* : Archivo donde se definen las configuraciones de tu aplicación.
- *Touch* : La copia de la carpeta descargada de *Sencha Touch*.
- *index.html* : una página inicial de *HTML* que incluye el archivo principal de *Sencha Touch*.

En la figura 15 se muestra un index muy básico de *Sencha Touch*

```
<!DOCTYPE html>
<html>
<head>
    <title>Iniciando con el desarrollo en Sencha Touch</title>
    <link rel="stylesheet" href="touch/resources/css/sencha-touch.css" type="text/css">
    <script type="text/javascript" src="touch/sencha-touch-all.js"></script>
    <script type="text/javascript" src="app.js"></script>
</head>
<body>
</html>
```

Figura 15: Ejemplo de HTML5 con Sencha Touch

Tras esto, creamos las interfaces gracias al archivo *app.js* comentado anteriormente. En la figura 16 observamos la función *TabPanel*, que crea una interfaz de paneles agrupados en *tabs* (en este caso mostramos sólo con uno) con el siguiente código:

```
Ext.application({
    name: 'Sencha',

    launch: function() {
        Ext.create("Ext.TabPanel", {
            fullscreen: true,
            items: [
                {
                    title: 'Home',
                    iconCls: 'home',
                    html: 'Welcome'
                }
            ]
        });
    }
});
```

Figura 16: Utilización de Javascript para Sencha Touch

El resultado de los dos códigos mostrados arriba da lugar, en un navegador con compatibilidad, aparece en la figura 17, donde se muestra el código sencillo que hemos implementado. Con la simple introducción de *CSS3* en estos ejemplos, podríamos variar la forma, color, fuente, etc, de los elementos que lo componen de manera muy sencilla.

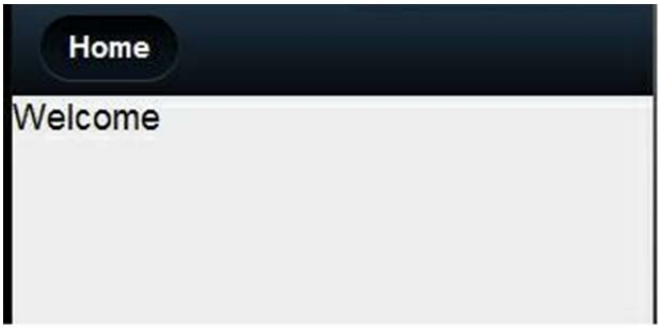

Figura 17: Visualización de los ejemplos en el navegador

III. III. EnyoJS

III. III. I Características

Este *framework* tiene como objetivo crear aplicaciones únicamente funcionales bajo el sistema operativo creado por *hp "webOS*".

Lo primero que se va a hacer es explicar qué es *webOS* y cómo surgió *enyoJS*

.

Lanzado por primera vez en 2009 con la *Palm*, un aparato predecesor del teléfono inteligente, *webOS* recibió buenas críticas por su intuitiva interfaz de usuario y sus herramientas para hacer múltiples tareas a la vez. Entre ellas una función que permitía a los usuarios ver las aplicaciones abiertas como una pila de cartas en la pantalla que se podían cerrar deslizando el dedo sobre ellas o poner en primer plano dándoles un toquecito. A pesar de las críticas favorables de los expertos, el software nunca ganó demasiado terreno entre los consumidores, ni siquiera después de que *Hewlett Packard* adquiriese una moribunda Palm por unos 1.380 millones de euros en 2010 e intentara resucitar sus productos móviles.

HP cerró su negocio de aparatos móviles, que incluía *webOS*, a finales de 2011. Pero en septiembre de este año la compañía lanzó Open *webOS*,su objetivo es poner a disposición de desarrolladores y fabricantes bajo licencia *Open Source* tanto el sistema operativo como el entorno de desarrollo del mismo, conocido bajo el nombre de **ENYO**.

El primer paso dado por Phoenix ha sido conseguir que *webOS* funcione como aplicación en teléfonos inteligentes ya existentes que operan con *Android.* (Otro grupo llamado *WebOS Ports* también está portando el software a otros aparatos como una tableta *Samsung Galaxy Nexus*).

El equipo de Phoenix ha construido una aplicación que sí funciona -aunque muy lentamente- en un teléfono inteligente Nexus S, y Zakutny afirma que esperan tener disponible una versión más rápida de la aplicación la tienda Google Play dentro de unos

dos meses (aunque no especifica cuánta de la funcionalidad del *webOS* original proporcionará la aplicación). El grupo quizá la ofrezca gratis con la esperanza de que hará que más gente se interese por *webOS*.

Cualquier aparato nuevo operando con *webOS* necesitará una nueva cosecha de aplicaciones. Phoenix espera resolver este tema usando *OpenMobile*, cuya tecnología hace que las aplicaciones *Android* puedan funcionar en aparatos que no tienen *Android* como sistema operativo. Al igual que *webOS*, *Open webOS* incorpora estándares de programación Web, cuya intención es facilitar a los desarrolladores Web crear aplicaciones o portarlas al sistema operativo.

Figura 18: Interfaz del sistema WebOS

EnyoJS trabaja con una biblioteca llamada *Onyx*, es una librería bastante completa, he aquí unos ejemplos:

Figura 19: Ejemplos de botones disponibles la librería Onyx

III. III. II Ventajas e Inconvenientes

A continuación se detallan una serie de ventajas e inconvenientes de este framework:

Ventajas

Todas las ventajas que aparecen a continuación están referidas a la versión 2.0 de libre código.

Al comenzar este PFC, esta versión no estaba disponible.

- Libre y de código abierto: 100% libre de utilizar, *Enyo* está disponible bajo la licencia Apache, versión 2.0.

- Extensible**:** con un pequeño y sólido núcleo, *Enyo* es modular y está diseñado para ser ampliado por la comunidad de desarrolladores.

- Construido para manejar la complejidad: modelo de componentes elegante *Enyo* hace que sea fácil de construir y mantener incluso las aplicaciones más complejas.

- Optimizado para móvil: *Enyo* tiene sus raíces en móvil y fue construido desde cero para brillar en las tabletas y los teléfonos.

- Ligero y Rápido: *Enyo* es pequeño (núcleo es <25k gzip) y ajustado para la velocidad y capacidad de respuesta en todas las plataformas soportadas.

Inconvenientes

- Aunque en la versión actual de *Enyo* se ha intentado que éste sea soportado por la mayoría de plataformas móviles y de escritorio, todavía queda un largo camino que recorrer.

- La información de *Enyo* se encuentra exclusivamente en inglés y al comienzo del PFC no había ninguna *Web App* que usara esta tecnología como ejemplo.

- Este framework está dirigido principalmente a *webOS,* un sistema operativo que como se ha comentado anteriormente está obsoleto a día de hoy.

III. III. III Compatibilidad

A pesar de la reciente liberación de código para hacer que este *framework* construya aplicaciones funcionales en todas las plataformas, al inicio del PFC esto no estaba asegurado, ni siquiera estaba disponible la versión gratuita actual.

En ese momento este *framework* se dedicaba a la elaboración de *Web Apps* destinadas al sistema operativo *webOS*, excuyendo el resto de sistemas operativos, entre los cuales se encuentran los más extendidos en el mercado como *Android, i0S y Windows Phone*.

Esto supuso una gran desventaja frente al resto de frameworks.

III. III. IV Ejemplo

En este apartado aparece el código de una aplicación web realizada con *EnyoJS* y su correspondiente visualización en el navegador.

```
<!DOCTYPE html>
▼<html class=" enyo-document-fit">
 ▶<head>…</head>
 ▼<body class=" enyo-body-fit webkitOverflowScrolling">
  ▼<div id="app" class="bg enyo-fit enyo-clip enyo-no-touch-action">
     <div class="overlap center bottom" id="app_bottom"></div>
    ▶<div style="display: none;" id="app_hats">…</div>
    ▶<div style="display: none;" id="app_sounds">…</div>
    ▶<div class="hcenter banner" id="app_title" style="width: 360px; height: 36px;">…</div>
     <canvas width="360" height="360" class="overlap hcenter" id="app_backgroundbox">
     <canvas width="360" height="360" class="overlap hcenter" id="app_selectbox">
     <canvas width="360" height="360" class="overlap hcenter border" id="app_hatbox">
   </div>
  </body>
 </html>
```

Figura 20: Código de una Web App usando biblioteca Canvas de Enyo

Figura 21: Visualización de la aplicación creada con EnyoJS

III. IV. Justificación de la elección del framework

Una vez vistos y estudiados detalladamente cada uno de los *frameworks* que podían suplir a *JQuery Mobile*, se ha llegado a la conclusión de que *EnyoJS* no nos servía para la creación de la aplicación web. Como se ha explicado en el estudio de los *frameworks*, en el inicio del PFC no estaba a disposición del usuario la versión de código abierto y la que había disponible además de ser de pago, sólo era útil en plataformas que usaran *webOs*, sistema operativo obsoleto en aquel momento.

Una vez descartado *EnyoJS*, quedan *LungoJS* y *Sencha Touch* como los dos *frameworks* más importantes para la creación de la aplicación web. Observando las características de cada uno, se llega a la conclusión de que ninguno de los dos es superior al otro, ya que los 2 cubren todo lo deseado para la creación del proyecto.

Es destacable que el procedimiento para elaborar la *Web App* será más laborioso con *Sencha Touch*, ya que tiene mucha más programación con *JavaScrip*t, lenguaje muy denso y complejo.

Una vez dicho todo esto, lo que ha desequilibrado la decisión entre los dos, además de que sea más costoso el desarrollo con *Sencha Touch*, ha sido el peso de la biblioteca, que es mucho menor en *LungoJS*, lo que hace que la aplicación tenga una velocidad muy superior.

Partiendo de que la elección de *Sencha Touch* como *framework* no hubiera sido una mala decisión, ni mucho menos, se ha creído que *LungoJS* es algo superior para el proyecto a desarrollar. Los argumentos principales para la elección de *LungoJS* son la mayor velocidad en el manejo de la aplicación y el premio a un producto nacional, que con pocos recursos, ha creado un producto tan bueno que hoy en día se encuentra en la élite de los *frameworks* empleados para el desarrollo de aplicaciones móviles.

IV. Elección del tema principal para la Web App

Tras habernos decantado por el empleo del *framework LungoJS* para crear nuestra aplicación, ahora llega el momento de decidir el tema central en el que se va a basar ésta.

Como se ha señalado en el *apartado II. IV,* nuestra aplicación va a girar en torno a la información contenida en una base de datos suministrada por el sitio web de *OpenData Navarra.*

Aunque en este proyecto se va a crear una aplicación de un tema concreto, el objetivo es crear una plantilla tipo que pueda servir para posteriormente crear cualquier otra aplicación que se desee, usando el desarrollador el tema que considere oportuno. Sólo será necesario aplicar una nueva base de datos en lugar de la utilizada en esta plantilla tipo.

Explotaremos todas las funcionalidades que encontremos de LungoJS, para que partiendo de nuestra plantilla, se ofrezca una amplia gama de posibilidades en la creación de sucesivas aplicaciones.

Dicho todo esto, en este proyecto tipo, se ha decidido que la aplicación gire en torno al tema de la educación (muy necesitada actualmente), concretamente en las **bibliotecas públicas de Navarra.**

Se va a emplear un archivo CSV alojado en OpenData Navarra, con información relevante de las bibliotecas de la comunidad como son: teléfono, dirección, horarios etc.

Con esta base de datos y el framework mencionado, se pretende realizar una aplicación sencilla y práctica para todos los usuarios.

V. Tecnologías empleadas

Después de haber explicado detalladamente cuáles son los conceptos básicos necesarios para comprender el proyecto y elegir un *framework* adecuado a las necesidades del mismo, vamos a mostrar las principales tecnologías que hemos empleado y para qué han servido.

Aquí mostramos las principales:

- *LungoJs*
- *HTML 5*
- *CSS3*
- *Javascript*
- *PHP*
- *Ajax*
- *Dreamweaver*
- *Filezilla*
- *OpenData Navarra*
- *Google Maps API*
- *Chrome Inspector*
- *Lungo Sugar*
- *Phpmailer*

Como muchas de ellas ya han sido explicadas en apartados anteriores, nos centraremos en las demás.

PHP

No se ha empleado demasiado en el proyecto, pero su utilidad es muy relevante. Este lenguaje, se ejecuta en el servidor, lo que significa que está oculto para el cliente, no lo ve. Lo único que sí ve el cliente es el resultado final.

Permite entre otras cosas, conectarse a bases de datos, enviar correos, realizar consultas etc. Cuenta además con la ventaja de ser compatible en todos los navegadores y puede formar parte de un archivo *HTML*

Ajax

Es una técnica creada para el desarrollo web. Permite crear aplicaciones interactivas que son ejecutadas en el lado del cliente, en su navegador, manteniendo la comunicación con el servidor en un segundo plano.

También hace que se puedan realizar modificaciones en la propia página sin tener que recargarla.

Dreamweaver

Es uno de los programas más empleados en los sectores del diseño y la programación web, gracias a su gran cantidad de funcionalidades.

Permite crear sitios de forma totalmente gráfica, y dispone de funciones para acceder al código *HTML* generado.

Según expertos, *Dreamweaver* es la herramienta más avanzada en el diseño de páginas web. El usuario que la maneje (aunque sea un experto programador de *HTML*) siempre encontrará razones para emplearla, sobre todo en lo que a productividad se refiere.

Cumple perfectamente el objetivo de diseñar páginas con aspecto profesional y soporta gran cantidad de tecnologías como:

- Hojas de estilo y capas
- *JavaScript* para crear efectos e interactividades
- Insertar archivos multimedia

En la figura 22 se puede observar como la imagen de la pantalla principal de *Dreamweaver*, donde a la izquierda aparece el código de la aplicación a desarrollar, en el centro se ve la aplicación que se está creando (ver cambios a medida que se realizan) y en el lado derecho aparecen menús para modificar la aplicación.

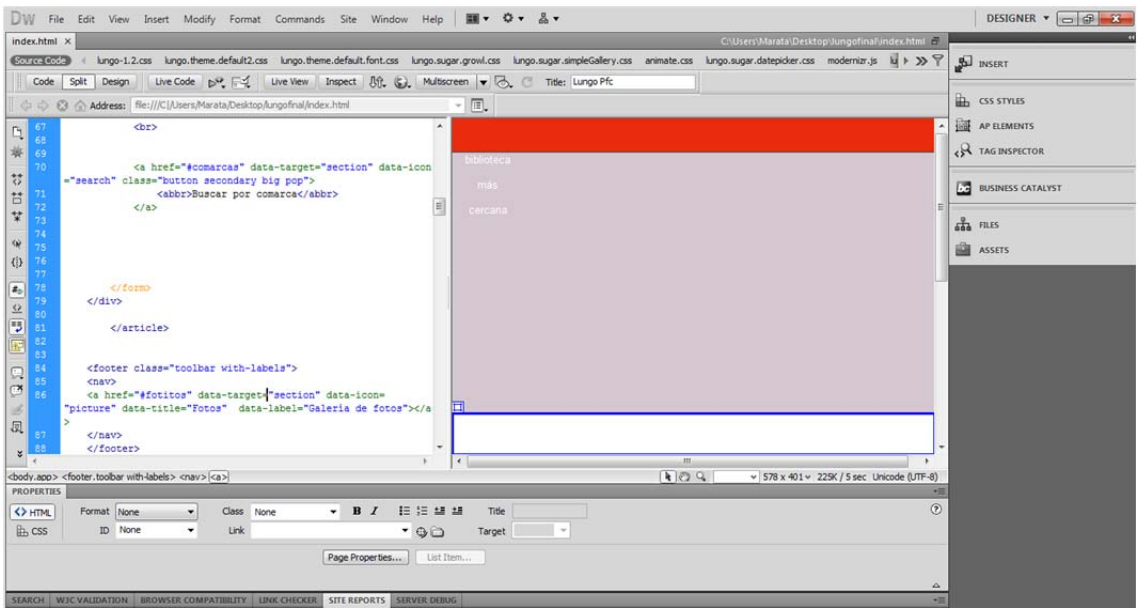

Figura 22: Ejemplo del programa Dreamweaver

Filezilla

Este programa se ha empleado para mantener una relación con el servidor en el que se encuentra alojada la aplicación, con todos sus archivos.

De esta manera se pueden realizar cambios en archivos que se encuentran en la memoria del ordenador para posteriormente introducirlos en el servidor web.

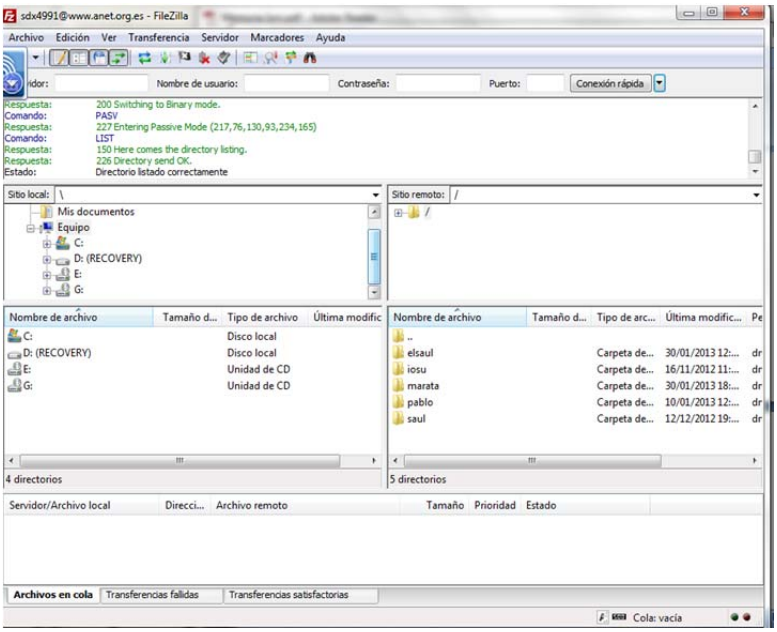

Figura 23: Ejemplo del programa Filezilla

Chrome Inspector

Es una herramienta integrada en el navegador *GoogleChrome* que permite observar el comportamiento interno de cualquier página web y ver sus errores de compilación o código, en caso de que existieran.

Ha sido muy útil en la realización del proyecto ya que, al detectar los fallos, es más fácil solucionarlos.

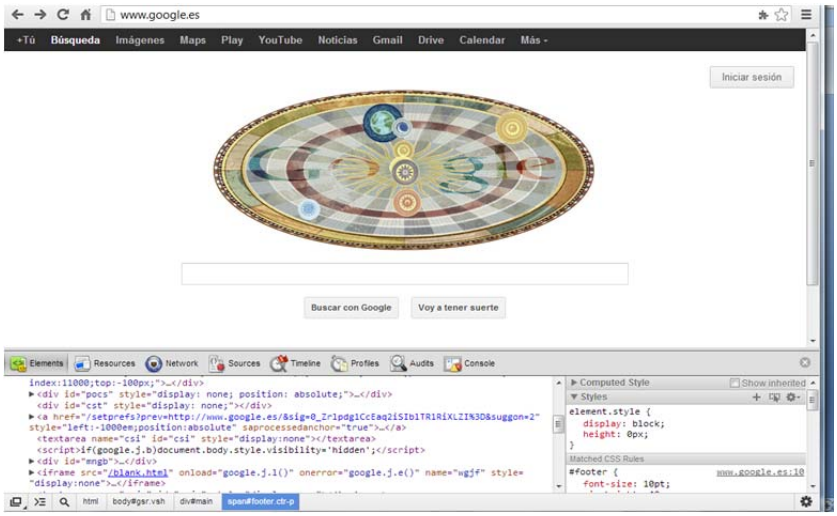

Figura 24: Ejemplo de la herramienta Google Inspector

LungoSugar

Conjunto de *plugins* que presenta *LungoJS*, creados por el propio autor del *framework*.

Estos enriquecen el marco de trabajo, añadiendo funcionalidades como calendarios, galerías de fotos o simples notificaciones animadas e interactivas.

Son archivos *CSS* y *JavaScript*.

PhpMailer

Complemento de *LungoJS* para que mediante un archivo *PHP* se puedan enviar correos electrónicos a una cuenta, desde el propio servidor en el que está alojada la aplicación

* No se ha hecho referencia a la *API de Google Map*s, ya que al ser una parte muy importante del proyecto, se explica en detalle dentro del desarrollo de la *Web App*.

VI. Desarrollo de la Web App

En este apartado se va explicar de forma detallada, el empleo de las tecnologías mencionadas anteriormente para la realización de nuestra *Web App*.

También se va a realizar una descripción completa de cada una de las partes que componen nuestra aplicación.

Debido a su gran importancia en este proyecto, vamos a mostrar claramente cómo se ha realizado la conexión con la base de datos descargada del portal web *OpenData Navarra*.

Por último se explicarán las funcionalidades que ofrece nuestra aplicación al usuario, mostrando ejemplos de cada una de ellas.

Como el tema de la navegación es vital en cualquier aplicación web, queremos incluir al final de este apartado una breve definición y una serie de diferencias entre los dos tipos principales de navegación: estática y dinámica.

VI. I. Estructura y maquetación

En primer lugar, vamos a mostrar en la figura 25 el esquema que define la estructura básica de nuestra aplicación:

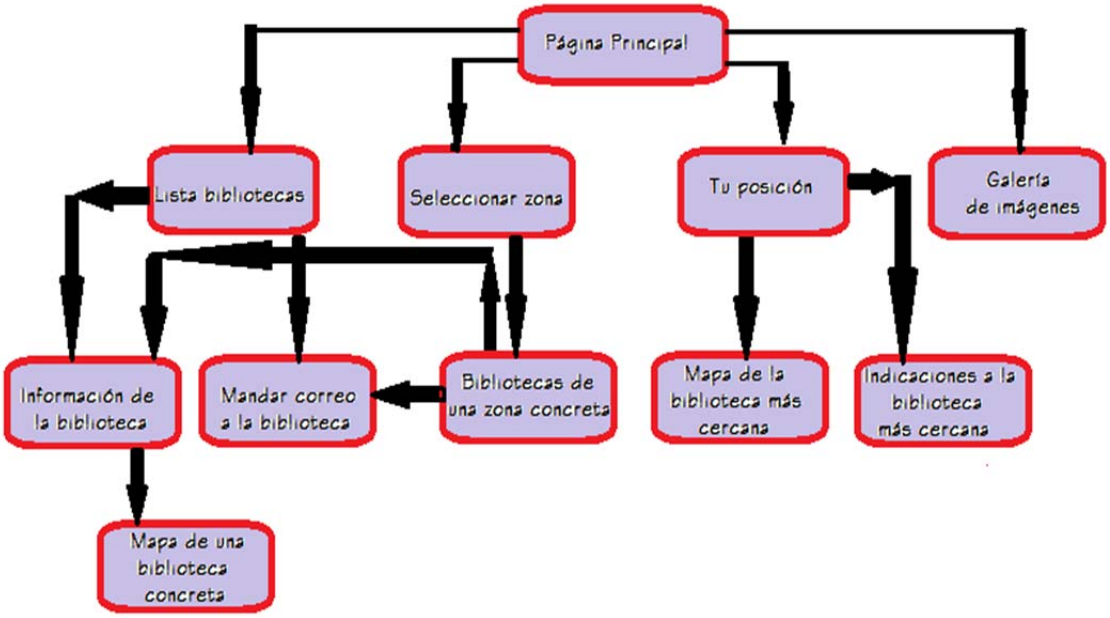

Figura 25: Estructura de la aplicación

Se puede apreciar en el esquema que se ha creado una sección principal, en la que podemos acceder a cuatro secciones: lista de todas las bibliotecas, menú para seleccionar una zona de Navarra, mapa con la posición del usuario y galería de imágenes.

En un segundo nivel, se encuentra la sección en la que colocamos la información de cada una de las bibliotecas. Dentro de esta sección aparecen dos artículos que seleccionaremos en el footer: uno donde aparece la propia información y otro para mandar un formulario de correo.

En un tercer nivel se ha creado una sección donde se mostrará en un mapa la posición de cada biblioteca. Para acceder hay que pulsar un botón (Ver mapa) dentro del artículo donde se ha alojado la información de la biblioteca.

Respecto a la sección de mapas, hemos introducido la opción de localizar la biblioteca más cercana y que en la misma sección se dispongan de dos artículos seleccionables en el footer, uno que mostrará en un mapa el camino hacia la biblioteca y otro en el que aparecerán las indicaciones a seguir para llegar a ésta.

Respecto a la maquetación, hay que dejar claro que la forma de las páginas se define mediante código HTML5. Se ordenan tanto las diferentes secciones, como los contenidos de éstas.

El diseño propiamente dicho, es decir, los colores de los botones, encabezados etc. y los tipos de letra y demás, se modifican en la propia hoja de estilos de LungoJS (CSS).

El propio framework incluye sus propias hojas de estilos predefinidas y si queremos realizar cualquier cambio debemos modificar el archivo '*lungo.theme.default2.css'*.

En este proyecto en concreto, sólo se han cambiado los colores de fondo de las listas y encabezados así como las letras de estos. Los bordes de las listas los hemos hecho desaparecer para lograr una mejor estética.

Se ha usado como color para el encabezado el rojo y las letras blancas en éste.

Para el texto de las listas se ha elegido un color de fuente negro y un color de fondo fucsia muy suave que encaja en el entorno gráfico de la aplicación

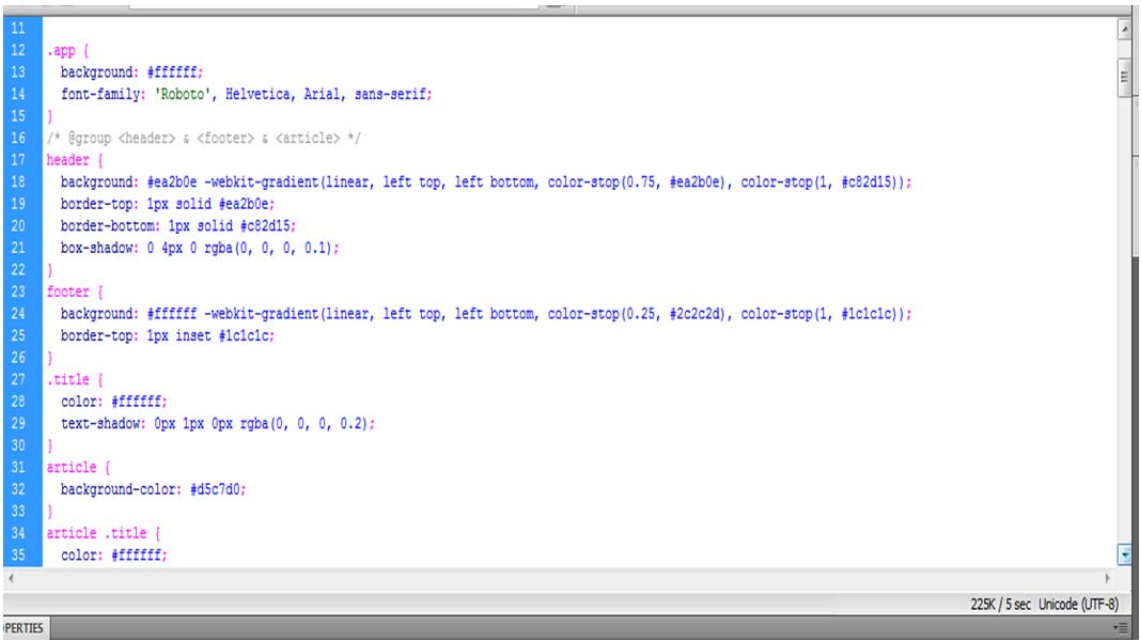

Figura 26: Ejemplo de código modificado en 'lungo.theme.default2.css'

VI. II. Conexión a los datos

Como se ha señalado anteriormente, nuestra aplicación se basa en unos datos recogidos de *OpenData Navarra,* por lo que son lo primero de lo que debemos disponer al empezar a realizar la *Web App*.

Una vez descargados desde la web esos datos en un archivo con formato CSV (Excel separado por comas), debemos almacenarlos en una base de datos *MySql*. Esta base de datos es gratuita y de libre uso y trabaja con un lenguaje propio, *Sql*.

Para acceder a ella y poder modificar su contenido, el profesor encargado del PFC nos suministró un nombre de usuario y una contraseña de acceso al programa online "*PhpMyAdmin*", que permite realizar esas modificaciones.

Para trabajar con la base de datos en este programa, hay que crear en primer lugar una tabla con los mismos campos que tenga ésta. Una vez hecho esto, se importa rellenando estos campos con la información de la base de datos.

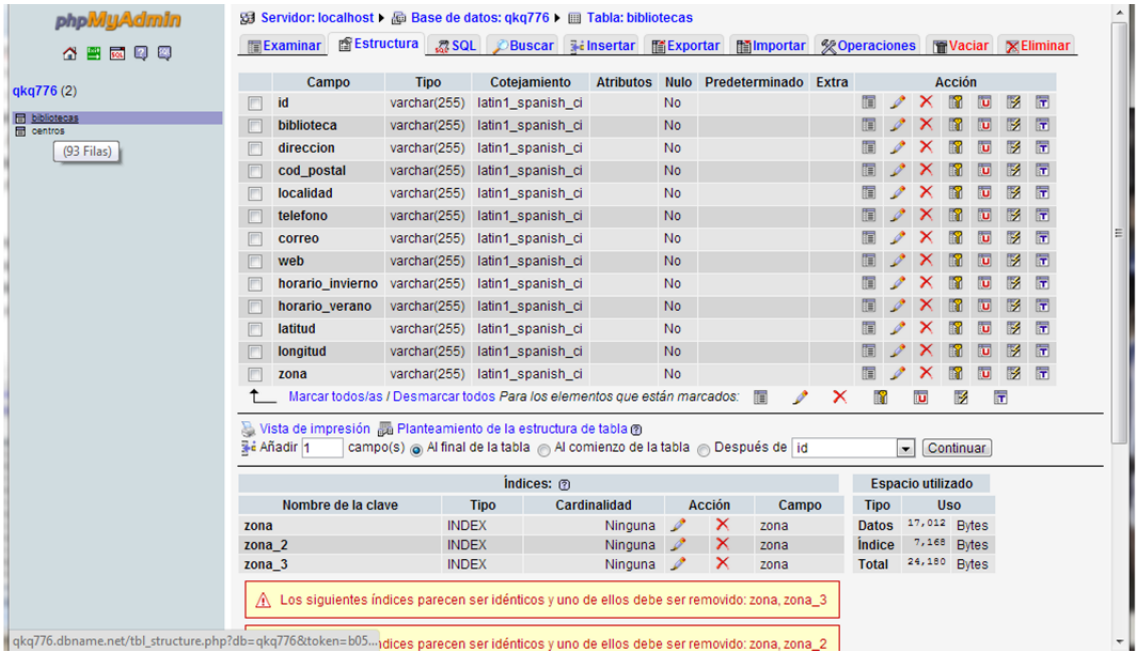

Figura 27: Ejemplo de base de datos en PhpMyAdmin

Tras esto, el siguiente paso es llamar a un PHP que permita conectarnos con esta base de datos. Esta llamada es necesaria ya que el PHP es ejecutado siempre en el lado del servidor, no se puede tener acceso a un servidor mediante HTML ya que nos encontramos en el lado del cliente y no del servidor.

Por lo tanto lo que se hace es llamar a este PHP para que se ejecute en el servidor, éste será el encargado de establecer conexión con la base de datos y nos devolverá esos datos en una tabla de valores. Este procedimiento debe crearse en el script "Services.js" (lenguaje JavaScript):

```
$$.json('mysql3.php',

{},

function(data) {

App.Data.cacherest(data);

});
```

Como ya se señaló anteriormente, a simple vista parece que no pasa nada ya que los archivos *Php* no se muestran en pantalla, sino que son ejecutados en el servidor, evitando la modificación en el aspecto de la página, lo cual es de agradecer.

En el propio archivo "mysql3.php" aparecen tanto la dirección donde se aloja la base de datos, como el usuario y la contraseña para acceder a ella, así como la tabla que se debe seleccionar y los datos que se deben recoger de ésta.

Para evitar el tener que conectarse con el servidor externo cada vez que el usuario realice una consulta de los datos, hemos creado una base de datos interna en LungoJS (en el script data.js) que se guarda en la caché de la página web.

De esta manera tenemos un gran ahorro de energía al evitando esta comunicación externa.

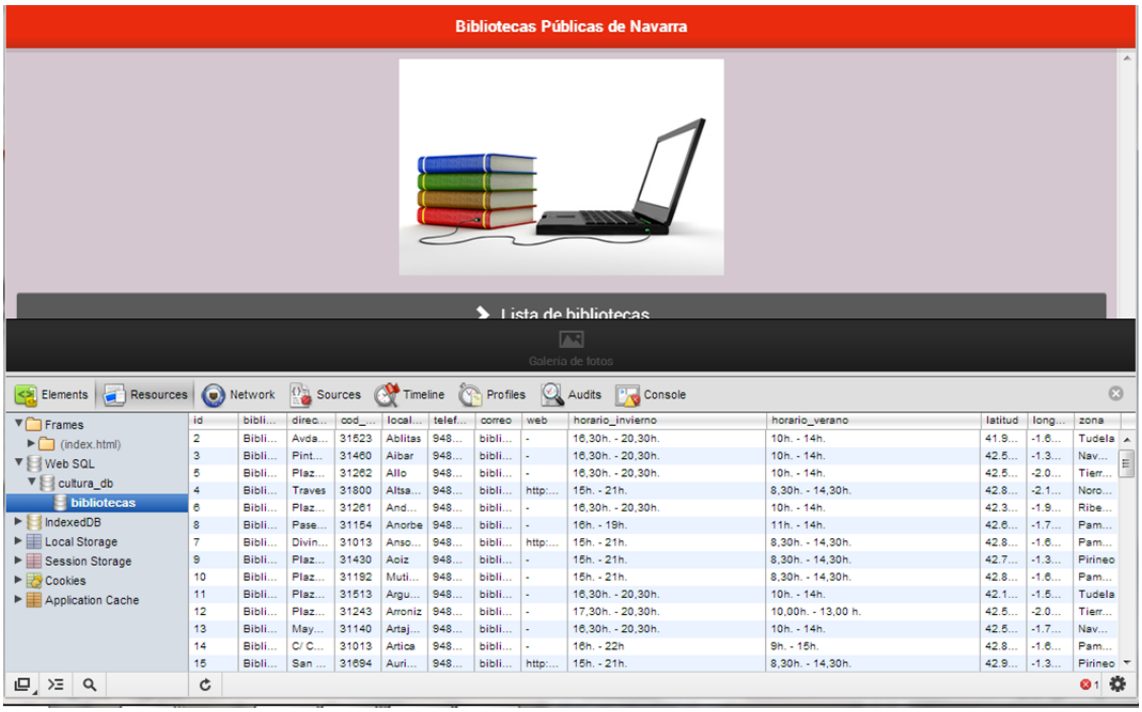

Figura 28: visualización con Chrome Inspector de la base de datos interna de LungoJS

Por último, se muestra un esquema sencillo para entender los procesos de petición, conexión y selección de la base de datos mediante el PHP:

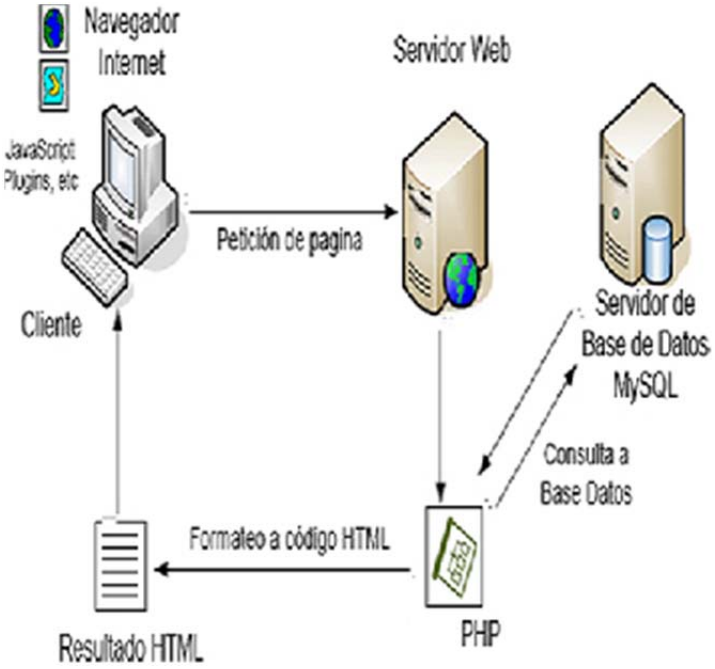

Figura 29: procesos de petición, conexión y selección de la BD mediante PHP

VI. III. Funcionalidades

En esta sección se van a explicar detalladamente los procesos seguido para la creación de cada una de las funcionalidades de la aplicación, con ejemplos sencillos para su fácil entendimiento.

VI. III. I. Mapas

Una de las herramientas más útiles de nuestra aplicación es, sin duda alguna, la de los mapas. En un principio intentamos implementar el mapa que venía por defecto en *LungoSugar, GMap*, pero tuvimos problemas de configuración y no funcionaba como queríamos. En este momento decidimos crear un nuevo script: "*map.js*" y aprovechar la API creada y desarrollada por *GoogleMaps* para este tipo de aplicaciones.

Esta herramienta desarrollada por la compañía Google, permite a diseñadores web incluir las funcionalidades de *GoogleMaps* y *GoogleStreetView* dentro de sus páginas de una forma clara y sencilla, sin apenas usar código para su creación.

Otra ventaja de trabajar con esta *API* es que es el propio google el que va a trabajar con los datos y además es reconocida en cualquier página HTML al pertenecer a Google.

Durante el desarrollo de la aplicación nos encontramos con un gran obstáculo en el camino, no disponíamos de la posición geográfica de las bibliotecas en la base de datos de *OpenData*, por lo que a partir de las direcciones de éstas tuvimos que introducir manualmente la longitud y latitud de cada biblioteca, lo que supuso una gran esfuerzo. Para futuras aplicaciones basadas en nuestra plantilla se recomienda el empleo de una base de datos que disponga de estas coordenadas, evitará bastante trabajo.

Dentro de nuestra aplicación, hemos usado *GoogleMaps* con dos propósitos bien diferenciados:

- Posicionar cada biblioteca en el mapa: para tener acceso a esta opción debemos dirigirnos a la sección donde aparece la información de la biblioteca y pinchar el botón (Ver mapa) que se encuentra dentro del apartado "Ubicación".
  Tras esto aparece en otra sección el mapa de la correspondiente biblioteca con su correspondiente indicador.
  Es el propio JavaScript de Lungo el que recoge la latitud y longitud de esa biblioteca (base de datos).

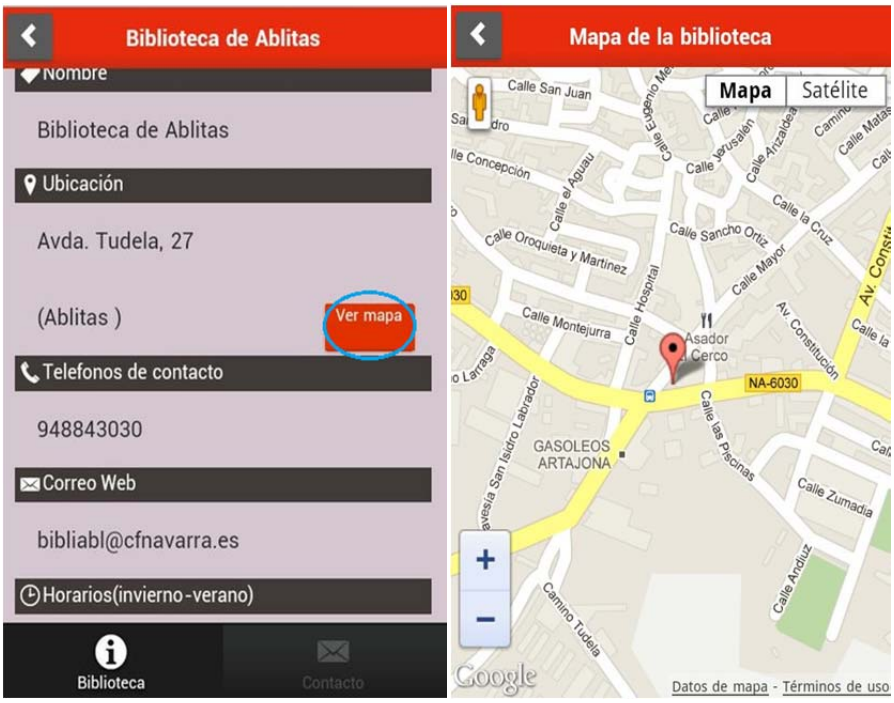

figura 30: Mapa de una biblioteca

- Posicionar al usuario en el mapa y encontrar biblioteca más cercana:

Para ver esta funcionalidad, se ha creado dentro de la página principal un botón que si pinchamos hace que aparezca en pantalla un mapa con un marcador, situando nuestra posición actual. Es lo que se denomina comúnmente geolocalización.

Figura 31: Geolocalización

En este mapa hemos creado otro botón en la parte superior (buscar biblioteca más cercana) que al pulsarlo crea otro marcador, el de la biblioteca más cercana y se muestran en el pie de la página dos opciones, ver el camino de tu posición actual a dicha biblioteca gráficamente y una serie de indicaciones de cómo ir.

Todo el aspecto técnico de este servicio se realiza mediante scripts, que recogen las posiciones de todas las bibliotecas, calculan la distancia desde el origen hasta ellas para finalmente elegir la más corta.

figura 32: Cómo ir a la biblioteca más cercana

Dentro del API de GoogleMaps, se han empleado las siguientes funciones:

- DistanceMatrix(): calcula las distancias desde la posición del usuario hasta los diferentes centros para finalmente quedarse con la mínima.

- DirectionsService():indica las direcciones a seguir hasta llegar al destino, es necesario introducir las coordenadas origen y destino (esta última será la posición extraída de *DistanceMatrix*)

- getCurrentPosition():encuentra la posición en la que se encuentra el usuario vía GPS, para lo cual tiene que estar habilitada esta opción en el aparato móvil, ya que hay algunos dispositivos que por seguridad la tienen deshabilitada.

Para entender y poder trabajar con estas funciones se ha necesitado estudiar previa y detalladamente la documentación de la *API* de *GoogleMaps*, ya que su lenguaje no tiene nada que ver con *LungoJS* al ser exclusivo de Google. Todo lo relacionado con estas herramientas que Google pone a disposición de los desarrolladores es libre y se puede encontrar en su página web:

https://developers.google.com/maps/documentation/javascript/

VI. III. II. Formulario de correo

Se ha creado un formulario de correo en nuestra aplicación para demostrar las posibilidades que ofrece LungoJS en este sentido.

Hay que aclarar que si se deseara hacer un uso comercial de esta aplicación, habría que tener acceso a cada uno de los correos de la biblioteca, para que cuando el usuario mande este formulario llegara a la biblioteca que él deseara.

Como no disponíamos de esta información (privada) lo que se ha propuesto es la creación de un formulario que sirva para verificar esta funcionalidad, aunque siempre mande el formulario a la misma dirección, la nuestra.

Simplemente se quería ver que *LungoJS* era capaz de mandar formularios, por lo que su diseño es simple pero conciso e intuitivo. Consta de 3 parámetros: nombre, dirección de correo y mensaje.

Figura 33: Formulario de correo de la aplicación

Para poder enviar este formulario, es necesario el uso de *PHP*, al no estar el navegador capacitado para ello. Es por esto, por lo que hay que habilitar un servidor especial de correo (SMTP) que se encargará de enviar el mensaje.

En primer lugar se crea una función dentro del script de LungoJS "Events.js", para que al realizar el usuario de la aplicación la acción de enviar, se recoja la información contenida en el formulario y se realice la petición mediante Ajax (JavaScript) al servidor.

En este proceso se usará el PHP que hemos creado (mail.php).

```
$$.ajax({
    type: 'POST', // defaults to 'GET'
    url: 'http://rest',
    data: {user: 'soyjavi', pass: 'twitter'},
    dataType: 'json', //'json', 'xml', 'html', or 'text'
    async: true,
    success: function(response) { ... },
    error: function(xhr, type) { ... }
});
```

Figura 34: Ejemplo del uso de Ajax con LungoJS

En el aspecto técnico, hay que señalar que por alguna razón que se desconoce, LungoJS ha implementado Ajax de tal manera que al realizar una petición de este tipo, el resultado sólo sea satisfactorio al ser aplicado en arrays, no en objetos.

Por esta razón, durante el proceso de desarrollo se han introducido todos los datos del formulario en un array, mediante una función sencilla.

Habría una forma de enviar el correo diferente a la que se ha elegido, mediante la función de PHP llamada mail (), pero tras haberlo intentado se llega a la conclusión de que su funcionamiento deja mucho que desear.

Por ello, se ha empleado un complemento: PhpMailer. Es una clase de Php para enviar emails que simplifica tareas complejas como enviar correos con ficheros adjuntos o en lenguaje HTML.

El modo de mandar correos más recomendable es con SMTP, aunque también puede emplearse Php mail() o sendmail. La ventaja del primero es que reparte la carga entre varios ordenadores, pudiendo enviar mayor cantidad de mensajes en un tiempo menor.

Dentro del PHP empleado se recogen los parámetros necesarios para el envío del mensaje: nombre, correo electrónico y mensaje del usuario. También se indica la dirección del servidor SMTP empleado y en caso de necesitarse el usuario y contraseña también se incluirán, tras esto se procederá al envío.

En conclusión, empleando el *plugin* PhpMailer y un servidor SMTP (previamente habilitado), se ha conseguido enviar correos desde nuestra aplicación.

Una característica importante de nuestro correo es que en caso de no rellenar correctamente alguno de los campos, aparecerá un mensaje indicando el fallo que ha surgido, dejando sin enviar el mensaje hasta que se solucione.

Una vez que todos los campos de nuestro correo hayan sido rellenados correctamente se deja enviar el correo y aparece una notificación de confirmación del envío.

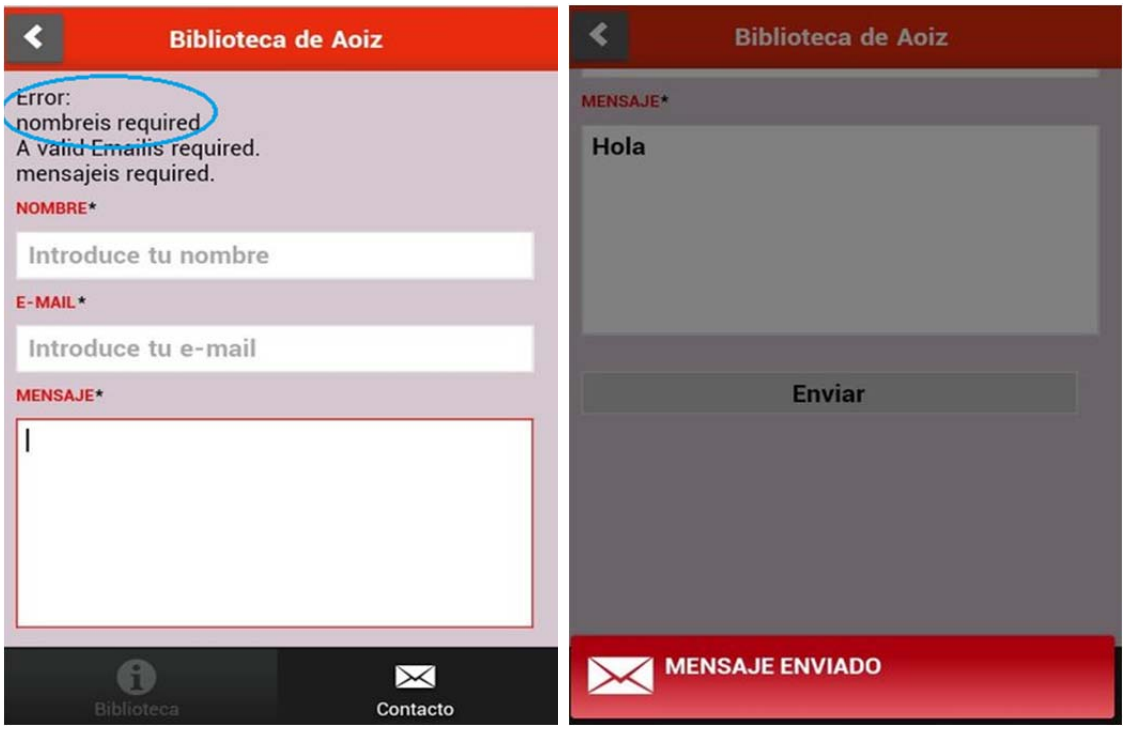

Figura 35: Correo con un campo incorrecto y correo correctamente enviado.

VI. III.III Galería de fotos

En primer lugar se quiere dejar claro que el motivo de la creación de una galería de fotos en esta aplicación sólo obedece a la necesidad de conocer y exprimir las funcionalidades de Lungo. No sería necesario implementar una galería de imágenes en una aplicación de este tipo.

Dicho esto, se quiere indicar que antes de la elaboración de la galería se realizó una búsqueda en la red de distintos tipos de galerías de Lungo, encontrando únicamente la empleada. En otros frameworks como jQuery el abanico de galerías de fotos para implementar es muchísimo mayor.

Para la implementación de la galería de la aplicación, acudimos al Sugar de Lungo (plugins para este framework) donde se encontró una galería muy sencilla: "SimpleGallery".

Se realizaron cambios en el código de este script, en vistas a que sólo se mostrase en la sección que nosotros deseáramos y que ocupara lo que quisiéramos. En concreto, para nuestra aplicación, el acceso a la galería se realiza a través del footer de la página principal.

Figura 36: Acceso a la galería de fotos desde el footer de la página principal

La galería creada consta de 7 fotos de las bibliotecas más importantes de Navarra y es muy sencilla.

El modo de interactuar entre ellas consiste en pinchar en la imagen que aparezca en pantalla para que aparezca la siguiente.

Figura 37: Galería de fotos de la aplicación

VI. IV. Navegación estática VS Navegación dinámica

Estos dos tipos de navegaciones formarán conjuntamente el motor para desplazarnos dentro de la aplicación.

Las dos interactúan simultáneamente y son claramente diferenciables

Navegación estática

En nuestra aplicación, la navegación estática se encarga de viajar entre distintas secciones. Estas secciones son fijas y se definen en la página *HTML* principal, siendo la navegación entre ellas previamente definida.

Un ejemplo sería, que al pinchar en un botón, por ejemplo de búsqueda, se acceda a otra sección, definida en el propio botón:

<a href="#seccion a la que queremos ir" data-target="section" class="button">

Nuestro objetivo es que la aplicación sea dinámica, es decir, que cuando se seleccione una de las bibliotecas, aparezca la información de dicha biblioteca en la correspondiente sección y que si se selecciona otra, aparezca la información de ésta en la misma sección.

Para ello se crean plantillas vacías con un esquema y unas características gráficas definidas (colores, iconos…), en las que se representarán los datos de la biblioteca seleccionada. En definitiva, lo que hacemos es definir una navegación entre diferentes plantillas vacías que serán rellenadas por el elemento (en este caso una biblioteca) elegido y seleccionado por el usuario.

Navegación dinámica

En nuestra aplicación, la navegación dinámica hace referencia a la parte de nuestra aplicación que hace que los contenidos varíen y se muestren en pantalla según lo solicite el usuario.

En este apartado juega un papel fundamental el lenguaje *JavaScrip*t dentro de *LungoJS* (*events, data, app, services* …). Será el encargado de mediante códigos específicos llevar a cabo esa variación, interactuando con la página *HTML* principal, que como se ha explicado anteriormente, se encarga de montar la plantilla rellenándola con datos. Son estos códigos de *JavaScript* los que definen los datos que deben emplearse en cada momento.

Se muestran a continuación cada una de las partes del framework elegido y su papel en este proceso:

- **Events.js:** es la parte que recoge las acciones que realiza el usuario, es decir, aquí se define cada elemento de la página (como un botón) y se indica que

cuando el usuario pulse dicho elemento se realice la llamada a una u otra función dependiendo de lo que se quiera.
A continuación aparece un un ejemplo de código en events.js para el botón Mapas de la página de inicio:

```
lng.dom('section#mapas  a[data-icon=search]').tap(function(event){
        App.Map.buscacerca();
         });
```

El funcionamiento es muy fácil de entender. Cuando se pulse el botón indicado, que se ejecute la función 'buscacerca' situada en Map.js.
Este es el esquema básico de events.js.

- **Data.js** en esta parte se trabaja con la base de datos *sql*. Este trabajo consiste en por ejemplo, recogerlos o modificarlos.
  Tras estos se llamará a otras funciones que los recogerán para sacarlos en pantalla o cualquier otro uso que se quiera hacer de ellos.

- **Map.js**: se encarga de todas las funciones relacionadas con GoogleMaps, como buscar la localización del usuario vía GPS, sacar un mapa en la pantalla o indicar una dirección definida mediante su longitud y latitud
  .
- **View.js**: define la forma final en la que se introducen esos datos, cómo aparecen en pantalla, indicando la zona exacta de la sección a la que se ha dirigido el usuario.

Esta estrategia de dedicar una plantilla vacía para todos los elementos de una tabla y rellenarla con código JavaScript, en lugar de crear una para cada uno, significará un enorme ahorro de espacio, lo que supondrá una mayor velocidad de navegación.

VII. Conclusiones

- En primer lugar queremos destacar que el framework *LungoJS* se promociona como un "fácil prototipo" para crear *Web Apps*, pero la realidad no es tan sencilla, hay que tener un cierto conocimiento previo tanto de *Javascript*, como de *Ajax* y *HTML5* si se quiere realizar unaW*eb App*. con unas funcionalidades básicas.

- En segundo lugar, quiero aclarar que el sistema *OpenData* dispone de muchas menos opciones de las que se presuponían en un principio. Las bases de datos que aporta no se actualizan en tiempo real, ni siquiera cada pocos días o semanas.
  La mayoría se crean y nunca se actualizan por lo que su principal objetivo, que es el de ofrecer información pública como el estado de las carreteras a tiempo real, no se cumple en absoluto.

- La elección de *LungoJS* como ya se ha comentado, ha sido porque se creía que su desarrollo era bastante asequible (aparte de otras razones), lo cual se ha podido confirmar a la hora de realizar el proyecto. Queda claro que con ciertos conocimientos previos, se puede formar una aplicación web potente, usando para ello unos scripts y código *HTML5*.
  En este sentido *LungoJS* ha cumplido con creces sus expectativas

- Respecto a otra de las razones por las que se ha decidido utilizar *LungoJS*, el "peso" de los archivos o cuánto tiempo tarda en cargar la *web app*, está claro que este es el mejor framework existente. Se ha comprobado el tiempo de navegación en ciertas aplicaciones creadas con *JQuery Mobile* y *Sencha Touch* y la conclusión es que *LungoJS* se encuentra un escalón por encima de ellas en este sentido.

- Este proyecto ha servido para que descubramos una gran puerta llena de posibilidades en el mercado laboral.
  Tanto la disposición pública de datos como el desarrollo de aplicaciones web están claramente en auge, por lo que la combinación de ambos puede suponer uno de los negocios más potentes a día de hoy.
  Tras una primera toma de contacto con estos sistemas, vamos a seguir buscando y aprendiendo nuevos métodos para que todo esto pueda ayudarnos en la introducción al mundo laboral, en una rama con grandes perspectivas de futuro.

- Esta última conclusión es un claro voto a favor de *LungoJS*. Es un gran motivo de elogio, el cómo una empresa como *QuoJs*, con muy poco presupuesto haya desarrollado uno de los más potentes *frameworks* para el desarrollo de *Web Apps*. en el mercado internacional.
  Estos chicos de Bilbao han dejado claro, que a veces las ganas de trabajar en un proyecto con las ideas claras puede ser mejor que invertir grandes cantidades de dinero en proyectos con una mala perspectiva de trabajo, como fue el caso del anteriormente mencionado sistema *WebOS*.

- Para visualizar la aplicación se debe acceder a la dirección:

  www.anet.org.es/pfcs/mobile/marata/lungo/index.html

VIII. Líneas futuras

En este apartado se va a mencionar una serie de mejoras que se podrían introducir tanto en el desarrollo del proyecto como en las herramientas empleadas en su elaboración.

He aquí las que se han considerado más relevantes o significativas:

- Se ha utilizado siempre la misma dirección de correo como destino de los formularios de contacto, pero una posible idea sería que cada biblioteca adjuntara su correo electrónico a la base de datos, de tal manera que el usuario de la aplicación pudiera ponerse en contacto con la biblioteca elegida

- Podría usarse un programa como 'PhoneGap' o 'Titanium Appcelerator' para convertir nuestra aplicación web a una aplicación nativa para dispositivos móviles y facilitar su accesibilidad a los usuarios.

- La galería de imágenes no sería necesaria en nuestro caso, pero si se deseara podrían recopilarse imágenes de las diferentes bibliotecas para añadirlos a la que se ha creado como prueba.

- El *framework LungoJs* necesita todavía unas cuantas mejoras ya que hemos encontrado bastantes problemas a la hora de utilizarlo. Por ejemplo el funcionamiento de las peticiones y servicios *Ajax* no están muy bien definidos y dan bastantes problemas, así como el funcionamiento de algunos elementos *scrollables* o la creación de plantillas para volcar los datos extraídos de la base de datos *Sql* interna, la cual es un punto a favor.

  Estamos seguros que todos estos problemas irán siendo solventados por los creadores de *LungoJS* que no paran de añadir mejoras a este framework en desarrollo.

IX. Bibliografía

- http://academica-e.unavarra.es/handle/2454/5734 → PFC Ion Iturri Gil

- www.google.es → buscador predeterminado de GoogleChrome

- www.navarra.es/home_es/Open-Data/ → OpenData Navarra

-lungo.tapquo.com → Página oficial del framework LungoJS

-www.sencha.com → Página oficial del framework Sencha Touch

-enyojs.com → Página oficial del framework *EnyoJS* para *WebOS*

- http://www.pidecurso.es/videos → Tutoriales/Screencasts de LungoJS

- github.com/TapQuo/Lungojs/commits → Repositorio público de LungoJS

- communiy.lungojs.com → Foro de usuarios de LungoJS

-qkq776.dbname.net → base de datos empleada en el PFC

- Manuales de lenguajes JavaScript, HTML5 y PHP → Varios autores